НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.056.55

«До захисту допущено»
Завідувач кафедри СПСКС

_____ В.П.Тарасенко_
(підпис) (ініціали, прізвище)
"___"_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
Комп'ютерні системи та компоненти

на тему: СПОСОБИ ГЕНЕРАЦІЇ КЛЮЧІВ АЛГОРИТМІВ ШИФРУВАННЯ НА БАЗІ ЛІНІЙНИХ СТРУКТУР

Виконав: студент II курсу, групи _КВ-71мп_____
(шифр групи)

Кісільчук Богдан Ярославович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник. доц., к.т.н. Тесленко О.К. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Спеціалізовані комп'ютерні системи

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
"___" _____ 2017 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Кісільчуку Богдану Ярославовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Способи генерації ключів алгоритмів шифрування на базі лінійних структур» науковий керівник дисертації доц.каф.СПСКС, к.т.н.,Тесленко О.К.

2. Термін подання студентом дисертації 7 грудня 2018 р.

3. Об'єкт дослідження: алгоритми шифрування на базі лінійних структур.

4. Предмет дослідження: способи генерації ключів алгоритмів шифрування на базі лінійних структур.

5. Перелік завдань, які потрібно розробити: проаналізувати швидкість алгоритму шифрування на базі лінійних структур з існуючими стандартами, розробити апаратну та програмну реалізацію даного алгоритму, проаналізувати та розробити алгоритми генерації секретних ключів.

6. Перелік ілюстративного матеріалу: презентація

7. Перелік публікацій: виступи на конференції та дві наукові конференції_____

8. Дата видачі завдання 1 грудня 2017 ._____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою дисертації	10.12.2017	
2.	Підготовка матеріалів першого розділу магістерської дисертації	21.02.2017	
3.	Підготовка матеріалів другого розділу магістерської дисертації	11.04.2018	
4.	Розробка апаратної реалізації алгоритму шифрування	05.05.2018	
5.	Розробка програмної реалізації алгоритму шифрування	07.06.2018	
6.	Підготовка матеріалів третього розділу магістерської дисертації	18.08.2018	
7.	Розробка алгоритмів генерації ключів шифрування	03.09.2018	
8.	Оформлення документації магістерської дисертації	14.10.2018	
9.	Попередній розгляд магістерської дисертації на кафедрі	26.11.2018	

Студент _____ Кісільчук Б.Я.
(підпис) (прізвище, ініціали)

Науковий керівник дисертації _____ Тесленко О.К,
(підпис) (прізвище, ініціали)

РЕФЕРАТ

Актуальність теми. Сьогодні існує велика потреба у забезпеченні шифрування інформації в режимі реального часу, але існуючі стандарти алгоритмів шифрування є дуже повільними для вирішення цієї задачі. Тому постає гостра необхідність у розробці алгоритмів шифрування, які могли це здійснювати на порядок швидше. В даній магістерській дисертації досліджуються способі генерації ключів алгоритмів шифрування на базі лінійних структур.

Об'єктом дослідження алгоритми шифрування на базі лінійних структур.

Предметом дослідження є способи генерації ключів алгоритмів шифрування на базі лінійних структур.

Мета роботи полягає у розробці способів генерації ключів алгоритмів шифрування на базі лінійних структур.

Методи дослідження. В даній роботі проведено аналіз існуючих алгоритмів шифрування, проаналізовано ключі алгоритму шифрування на базі лінійних структур, їх вразливості та недоліки, розроблені способи генерації ключів.

Наукова новизна роботи полягає у подальшому розвитку алгоритмів шифрування на базі лінійних структур та способів генерації секретних даних алгоритмів (ключів).

Практична цінність отриманих в роботі результатів полягає в тому, що розроблені алгоритми шифрування на базі лінійних структур є швидшими за існуючі стандарти і можуть використовуватися у тих випадках, де необхідно забезпечити шифрування в режимі реального часу.

Апробація роботи. Результати роботи пройшли апробацію на наукових конференціях:

– XI конференція молодих вчених «Прикладна математика та комп’ютинг» ПМК-2018-2;

– XX міжнародна науково-технічна конференція SAIT 2018.

Структура та обсяг роботи. *Магістерська дисертація складається з вступу, трьох розділів, висновків та додатків.*

У вступі охарактеризовано стан проблеми, обґрунтовано актуальність напрямку досліджень, визначено завдання дослідження, описано виконану роботу.

У першому розділі виконано аналіз сучасних стандартів шифрування та виконано порівняння швидкості стандартів з алгоритмом на базі лінійних структур.

У другому розділі розроблено можливі апаратні, на базі ПЛІС, та програмні реалізації алгоритму на базі лінійних структур.

У третьому розділі сформульовано три можливих варіанти секретних ключів, проведено теоретичний аналіз криптоатак на ключі та розроблено модифікації до алгоритмів генерації ключів.

У висновках підведено підсумок зроблених досліджень.

У додатках наведено фрагменти програмного коду, що реалізує алгоритми генерації ключів, копії публікацій.

Робота виконана на __ аркушах, містить__ додатків та посилання на список використаних літературних джерел з __ найменувань. У роботі наведено __ рисунків та __ таблиць.

Ключові слова: алгоритми шифрування, способи генерації ключів алгоритмів шифрування, алгоритми шифрування на базі лінійних структур, ключі алгоритмів шифрування.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

5G – п'яте покоління мобільного зв'язку, що діє на основі стандартів телекомунікацій, наступних за існуючими стандартами 4G / IMT-Advanced.

Дешифрування – процес несанкціонованого отримання інформації з зашифрованих даних. При цьому ключ дешифрування зазвичай невідомий.

КМ (конструктивний модуль)

Криптоаналітик – спеціаліст по криптоаналізу.

ОККМ (одновимірний каскад конструктивних модулів)

ПЛІС (програмована логічна інтегральна схема) – електронний компонент, що використовується для створення цифрових інтегральних схем.

Розшифрування – процес санкціонованого перетворення зашифрованих даних у придатні для читання.

Шифрування – алгоритмічне (криптографічне) перетворення даних, яке виконується у посимвольній послідовності з метою одержання шифрованого тексту.

AES (Advanced Encryption Standard) – симетричний блочний шифр, вибраний урядом США для захисту секретної інформації, що реалізується програмним та апаратним забезпеченням у всьому світі для шифрування конфіденційних даних.

CPLD (Complex Programmable Logic Device) – програмована логічна інтегральна схема в діапазоні складності між мікросхемами PAL і FPGA, що поєднує їх архітектурні рішення.

DES (Data Encryption Standard) – це симетричний ключовий блочний шифр, опублікований Національним інститутом стандартів і технологій (NIST).

FPGA (Field-Programmable Gate Array) – інтегральна схема, призначена для налаштування користувачем або розробником після виготовлення.

LUT (Look-Up Table) - це структура даних, зазвичай масив або асоціативний масив, що використовується з метою замінити обчислення на операцію простого пошуку.

RSA (Rivest–Shamir–Adleman) – криптографічний алгоритм з відкритим ключем, що базується на обчислювальній складності задачі факторизації великих цілих чисел.

SSL(Secure Sockets Layer) – криптографічний протокол, який забезпечує встановлення безпечного з'єднання між клієнтом і сервером. SSL спочатку розроблений компанією Netscape Communications.

ВСТУП

В сьогоднішньому світі інформація є одним із найважливіших ресурсів. Всім відоме висловлювання Натана Ротшильда: «Хто володіє інформацією, той володіє світом» [1], - і справді, наявність інформації у однієї людини та її відсутність у іншої, може відігравати ключову роль в успішній реалізації планів першої людини. Саме необхідність у зменшенні ризику отримання такої інформації сторонніми особами призвела до збільшення потреби в захисті інформації.

Сьогодні існує велика кількість різноманітних методів захисту від несанкціонованого доступу, але особливе місце серед них займають криптографічні методи. Оскільки, на відміну від інших методів, вони опираються тільки на властивості інформації, а не на властивості передачі, зберігання інформації або апаратні особливості.

В наш час існує дуже велика потреба обміну інформацією в режимі реального часу, наприклад при керуванні безпілотниками. Або ж передача великих об’ємів даних, наприклад, за допомогою мобільних мереж. Адже незабаром планується введення нового телекомунікаційного стандарту 5G [2], що дозволить збільшить швидкість завантаження даних до 100Мбіт/сек. Але хоча сьогодні існує велика кількість алгоритмів шифрування, проте сучасні стандарти шифрування є досить повільними, оскільки виконують велику кількість операцій під час шифрування. Таким чином виникає необхідність у розробці алгоритму шифрування, який міг би здійснювати це на порядок швидше.

В даній магістерській дисертації проаналізовано та проведено порівняння швидкості існуючих стандартів алгоритмів шифруванні, генерацію ключів в них та описано принцип роботи алгоритму шифрування на базі підстановок довільної розрядності. А також досліджено апаратну реалізацію алгоритму на базі ПЛІС, його програмну реалізацію та можливі

модифікації до нього, наприклад, використання суперпозиції підстановок або зміна розрядності підстановок.. Проаналізовано можливі криптоатаки на секретний ключ алгоритму та відповідно до цього визначено, які дані алгоритму можна віднести до секретних. Досліджено можливі способи генерації ключів алгоритму та розроблено модифікації до алгоритмів генерації.

Дослідження алгоритму шифрування на базі лінійних структур має дуже велику актуальність, оскільки даний алгоритм є на порядок швидшим за прийняті стандарти (Калина, AES), а його апаратна і програмна реалізації є дуже простими, за рахунок того, що він базується на регулярних структурах.

РОЗДІЛ 1
АНАЛІЗ ТА ПОРІВНЯННЯ ІСНУЮЧИХ АЛГОРИТМІВ ШИФРУВАННЯ

1.1 Перші алгоритми шифрування

Перші згадки про використання шифрування датуються числами до нашої ери. Це означає, що вже навіть у ті часи, у людей була необхідність засекречувати певним чином інформацію. Одним із перших таких алгоритмів шифрування був шифр Цезаря. Даний шифр є типовим прикладом шифру підстановки, оскільки, суть його полягає в наступному, кожен символ вхідного тексту заміняється іншим, що знаходиться по відношенню до першого, на кілька позицій правіше. В такому разі, якщо індексом кожного символу в алфавіті буде його порядковий номер, то шифрування можна описати наступною формулою:

$$y = (x + k)\ mod\ n,$$

де x – символ вхідного тексту, y – символ зашифрованого тексту, n – кількість символів в алфавіті, k – ключ.

Відповідно формула для дешифрування наступна:

$$x = (y - k)\ mod\ n$$ [3].

Даний шифр є дуже ненадійним, оскільки дешифрувати його можна, знаючи лише зашифрований текст. Якщо криптоаналітик знає, що текст зашифровано шифром Цезаря, то дешифрувати такий текст, можна за допомогою методу повного перебору, оскільки існує не так багато варіантів зсувів, для англійської мови це число дорівнює 26, а для української – 33. Якщо ж криптоаналітик не знає, що даний текст зашифровано за допомогою шифром Цезаря, тоді за допомогою методу частотного аналізу, можна з’ясувати, що даний текст зашифровано шифром Цезаря, і дешифрувати його, як було сказано раніше, методом повного перебору. Суть частотного аналізу полягає в наступному. Вважається, що ймовірність появи певної букви, в

досить довгому тексті є визначеною, в такому разі, якщо в зашифрованому тексті, є символ, який має таку ж ймовірність, то можна припустити, що даний символ є відповідним зашифрованим символом вхідного тексту.

1.2 Типи алгоритмів шифрування

Як наука криптографія з'явилась після фундаментальних робіт американського математика та електротехніка Клода Шеннона. В його роботах «Математична теорія зв'язку» та «Теорія зв'язку в секретних системах» міститься узагальнення велетенського досвіду створення шифрів, що був накопичений ще до нього, та розробляється математичний апарат для криптографічних задач.

Аналізуючи шифри, що існували раніше, Клод Шеннон прийшов до висновку, що більшість з них сконструйовані з типових компонент, що здійснюють заміну та перестановку. Більш глибоке розуміння того, як повинні будувати надійні шифри, дозволило йому виділити двох головних принципів побудови криптографічних перетворень: перемішування та розсіювання. Таким чином Клод Шеннон був основоположником симетричного шифрування, яке ми сьогодні знаємо [4].

Алгоритми шифрування поділяються на 2 типи:

1. симетричні;

2. асиметричні.

Симетричне шифрування - це стандартний метод шифрування. Це також найпростіший з двох типів шифрування. Симетричне шифрування виконується за допомогою лише одного секретного ключа, який називають "симетричним ключем", яким володіють обидві сторони. Цей ключ застосовується для шифрування та розшифрування інформації. Відправник використовує цей ключ перед надсиланням повідомлення, і отримувач використовує його для розшифровки повідомлення (рис 1.1) [5].

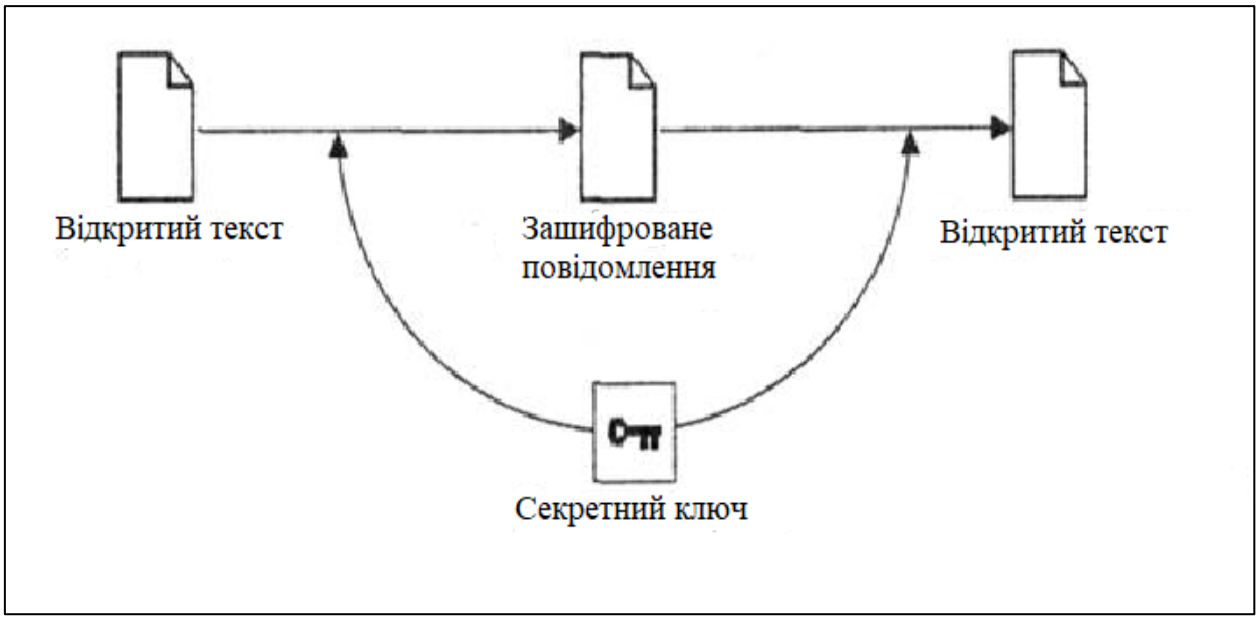

Рисунок 1.1 – Схема симетричного шифрування.

Це досить таки простий спосіб, і, як наслідок, це не займає багато часу. Коли справа доходить до передачі великих об’ємів даних, симетричне шифрування є кращим вибором. Шифр Цезаря - це хороший приклад симетричного шифрування. Сучасні підходи симетричного шифрування виконуються з використанням таких алгоритмів, як RC4, AES, «Калина», DES, 3DES, QUAD, Blowfish та ін.

Найпоширеніша форма симетричного шифрування відбувається після того, як зашифроване з'єднання було встановлене між клієнтом і сервером із встановленим сертифікатом SSL. Після підключення створюються і обмінюються два 256-бітні сеансові ключі, таким чином може бути утворено зашифроване з’єднання.

На відміну від симетричних алгоритмів шифрування, де для шифрування та розшифрування використовується один і той же ключ, в асиметричних це робиться за допомогою двох різних ключів (рис. 1.2).

Ключ, за допомогою якого відбувається шифрування якогось повідомлення, називається публічним, оскільки він передається у відкритому вигляді і хто завгодно може його побачити.

Ключ, за допомогою якого відбувається розшифрування, називається закритим або особистим ключем, оскільки він повинен бути відомим лише отримувачу.

Головною особливістю такого типу алгоритмів є те, що за допомогою відкритого ключа неможливо розшифрувати зашифроване повідомлення, це може зробити лише приймаюча сторона за допомогою згенерованого закритого ключа. Саме тому для того щоб передати ключ, яким шифрується повідомлення не потрібно ніяких спеціальних засобів [5].

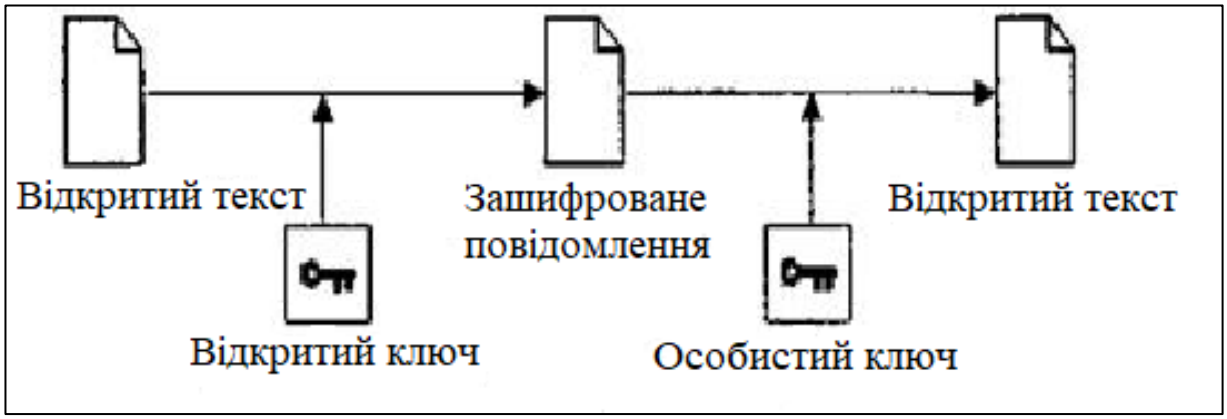

Рисунок 1.2 – Схема асиметричного шифрування.

1.2.1 Симетричні алгоритми шифрування

Симетричні алгоритми шифрування поділяються на:

1. потокові;
2. блочні.

До потокових алгоритмів шифрування відносяться алгоритми які дозволяють шифрувати неперервні потоки інформації. Схема їх роботи наступна, алгоритм генерує значення біту випадковим чином, після цього на наступний біт вхідного потоку інформації накладається значення згенерованого біту, за допомогою операції XOR (рис. 1.2) [6].

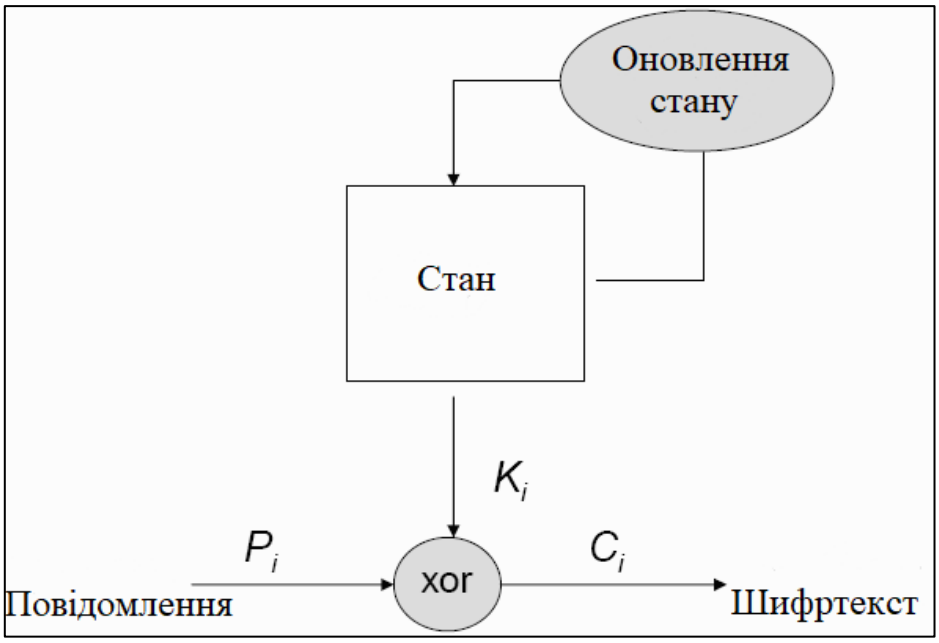

Рисунок 1.3 – Схема роботи потокових алгоритмів шифрування.

Блочні алгоритми шифрування в свою чергу розбивають вхідне повідомлення на блоки розміром в кілька байт і шифрують вже дані блоки інформації. В результаті на виході отримуються зашифровані блоки такого ж розміру, як і були до шифрування, і відповідно, оскільки це симетричні алгоритми шифрування, для того щоб розшифрувати необхідно використати той же ж секретний ключ, що і для шифрування [7].

Одним з прикладів симетричних блочних алгоритмів є DES (Data Encryption Standard), що був затверджений владою США, як стандарт таких алгоритмів у 1977 році.

В своїй реалізації DES використовує блоки розміром 64 біта, ключ розміром 56 біт та мережу Фейстеля. На виході отримується зашифрований блок тексту, такого ж розміру як і вхідний блок, при чому, якщо змінити хоч 1 біт блоку вхідного повідомлення на виході буде отримано абсолютно інший зашифрований блок. Процес шифрування складається з двох перестановок та 16 раундів Фейстеля (рис 1.3). Кожен раунд використовує різні згенеровані ключ розміром 48 біт [8].

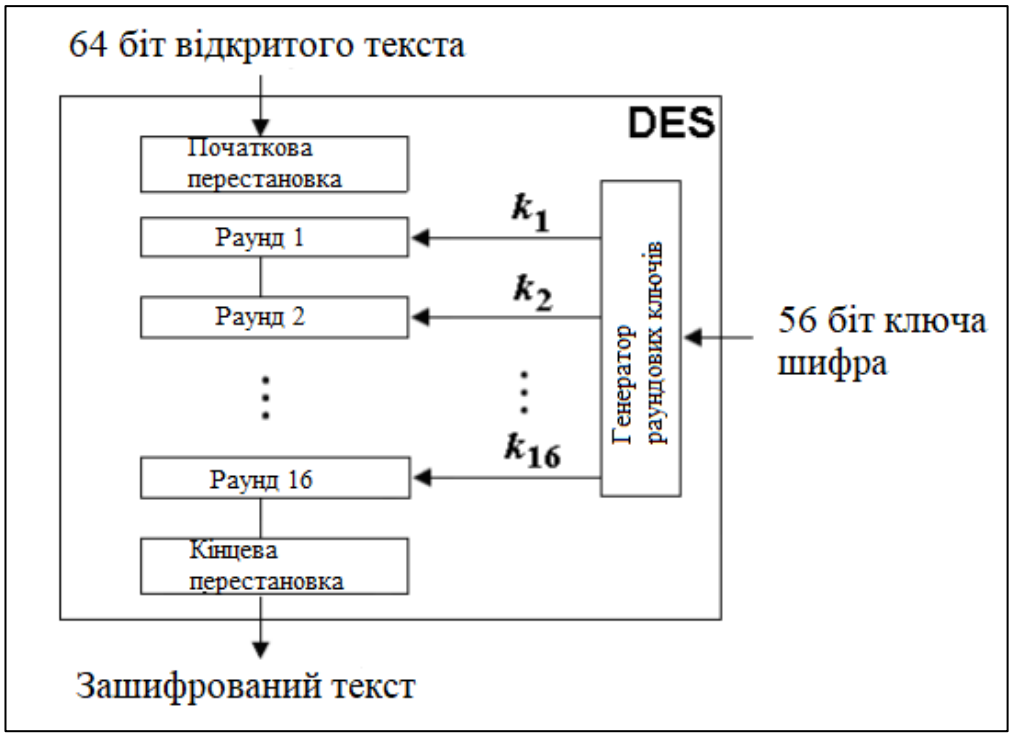

Рисунок 1.4 – Схема роботи алгоритму DES.

На вхід кожної перестановки поступає 64 біта, які потім переставляються згідно заданої таблиці. Ці перестановки є взаємо оберненими. Обидві перестановки не мають ніякого значення з точки зору криптографії, тому досі не відомо для чого їх додали в DES проектувальники алгоритму [9].

DES виконує 16 раундів, які використовують шифр Фейстеля (рис. 1.4).

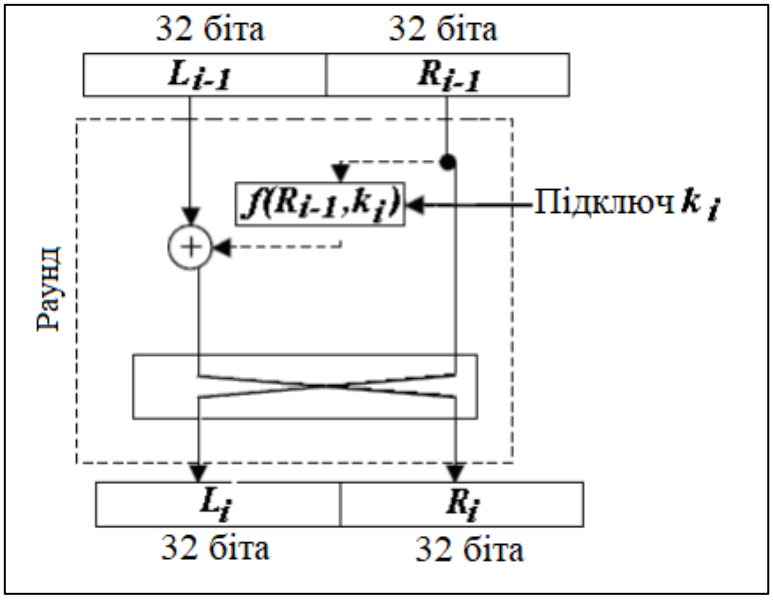

Рисунок 1.5 – Раунд DES.

Кожен раунд приймає на вхід напівблоки L_{i-1} і R_{i-1} з попереднього раунду або початкового блоку перестановки і створює напівблоки L_i та R_i для входу в наступний раунд або кінцевий блок перестановки. Всі необоротні елементи зосереджені в функції $f(R_{i-1}, k_i)$.

Функція DES за допомогою 48-бітного ключа зашифровує 32 біти правого напівблока R_{i-1} і на виході отримується 32-бітний результат. Дана функція складається з 4 складових: Р-бокс розширення, операція XOR, група S-боксів та прямий Р-бокс (рис. 1.5).

Р-бокс розширення слугує для розширення вхідного блоку розміром 32 біти до 48 біт, для того щоб узгодити його розмір з розміром раундового підключа. Для цього вхідний блок розбивається на 8 секцій по 4 біти і кожна така секція розширюється до 6 біт.

Після розширення DES використовує операцію XOR над розширеною частиною напівблока і раундовим ключем k_i.

Після цього блок розміром 48 біт ділиться на 8 послідовних 6-бітних векторів, кожен з яких заміняється на 4-бітний вектор за допомогою S-боксів.

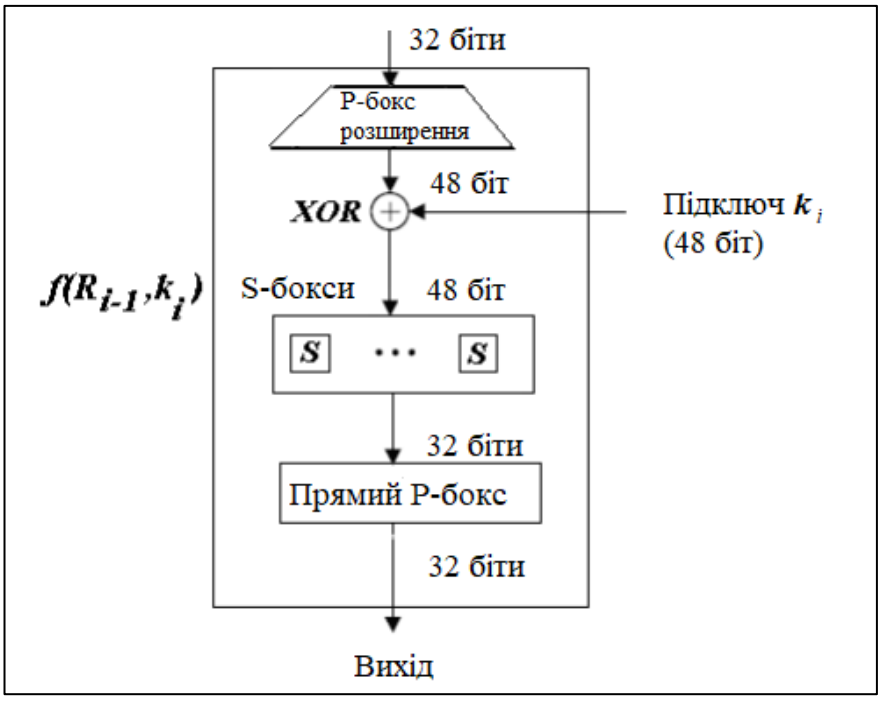

Рисунок 1.6 – Функція DES.

Далі біти блока перемішуються в прямому P-боксі на основі заданої таблиці.

Після 16 раунду правий і лівий блоки не міняються місцями, а об'єднуються в блок $R_{16}L_{16}$ і піддаються фінальній перестановці IP^{-1} [8].

DES створює 16 раундових ключів k_i по 48 біт із ключа k шифру на 56 біт. Але для того, щоб задати ключ шифру треба серед 56 біт ключа додатково вписати ще 8 біт на позиції 8,16,…,64 для перевірки парності таким чином, щоб кожен байт містив непару кількість одиниць. За допомогою цієї операції виявляються помилки при обміні та збереженні ключі.

Генерація ключів складається з наступних етапів:

1. Перестановка стиснення для видалення біт перевірки – із 64-бітного ключа видаляються біти 8,16,…,64 і переставляються біти згідно з таблицею (рис. 1.6).

57	49	41	33	25	17	9	1	58	50	42	34	26	18	C_0
10	2	59	51	43	35	27	19	11	3	60	52	44	36	
63	55	47	39	31	23	15	7	62	54	46	38	30	22	D_0
14	6	61	53	45	37	29	21	13	5	28	20	12	4	

Рисунок 1.7 – Таблиця перестановки бітів в ключі шифру

2. Після перестановки 56 бітів ключа діляться на блоки C_0 і D_0 по 28 біт кожен. Потім для генерації раундових ключів із блоків C_0 і D_0 за допомогою операції циклічного зсуву вліво на 1-2 біти формуються блоки C_i і D_i, $i=1,2,...16$. В раундах 1,2,9 і 16 зсув відбувається на 1 біт, а у всіх інших раундах – на 2 біти. Після визначення блоків C_i і D_i біти цих блоків об'єднуються в один ключа на 56 біт.

3. Перестановка стиснення згідно з відповідною таблицею (рис. 1.7) змінює 56 біт на 48 біт, які формують раундовий ключ [10].

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Рисунок 1.8 – Таблиця перестановки стиснення ключа

При розшифруванні – раундові ключі ті ж самі, що і при шифруванні, але тепер вони використовуються в оберненому порядку.

Головною перевагою даного алгоритму є те, що шифрування та розшифрування відбувається за однаковим алгоритмом, що є дуже важливим параметром при апаратній реалізації.

Недоліками ж є:

- Малий розмір ключа. Розмір ключа дорівнює лише 56 біт, що при сучасних обчислювальних можливостях є дуже малим значенням. Оскільки за допомогою використання паралельних обчислень даний ключ можна зламати за допомогою силової атаки методом повного перебору.
- Слабкі ключі. Раундові ключі є проблемою, оскільки під час ділення ключів на 2 частини і зміні їх місцями на виході може бути однаковий результати, якщо вони мають безперервні 1 або 0. Таким чином один і той же ключ, використовується у всіх раундах;
- Може бути однаковий результат на виході з S-боксів при різних вхідних значення на перестановці;

Але оскільки розмір ключа в DES є досить невеликим, а це означає що підібрати ключ можна за досить короткий час, то в 1978 році було створено алгоритм 3DES, який базується на звичайному DES. Суть 3DES полягає в

тому, що три рази виконується DES і відповідно розмір ключа збільшується в три рази, але через це значно знижується швидкість шифрування.

В результаті в 2002 році було затверджено новий стандарт AES (Advanced Encryption Standard). AES – це симетричний ітеративний блочний алгоритм, на відміну від DES він не використовує мережу Фейстеля. Розмір блоку в даному алгоритмі дорівнює 128 біт, а розмір ключа може бути 128, 192, 256 біт.

AES представляє вхідний блок даних у вигляді масиву байтів розміром 4x4, що називається станом. Наприклад, якщо є 16 байт $a_0, a_1, \dots, a_{15}$, то в двовимірному масиві вони будуть виглядати наступним чином:

$$\begin{bmatrix} a_0 & a_5 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

Розмір ключа, який використовується для шифру AES, визначає кількість раундів перетворення, які перетворюють вхідне повідомлення у зашифроване. Кількість раундів виглядає наступним чином:

- 10 раундів для 128-бітних ключів.
- 12 раундів для 192-бітних ключів.
- 14 раундів для 256-бітних ключів.

Перед першим раундом виконується додавання по модулю 2 вхідного блоку з початковим ключем шифру.

Раунди даного алгоритму виконуються за наступною схемою (рис. 1.8):

1. таблична заміна кожного байту масиву даних;
2. циклічний зсув вліво всіх рядків вхідного масиву, крім 0;
3. перемішування байт в колонках;
4. операція XOR з ключем раунду.

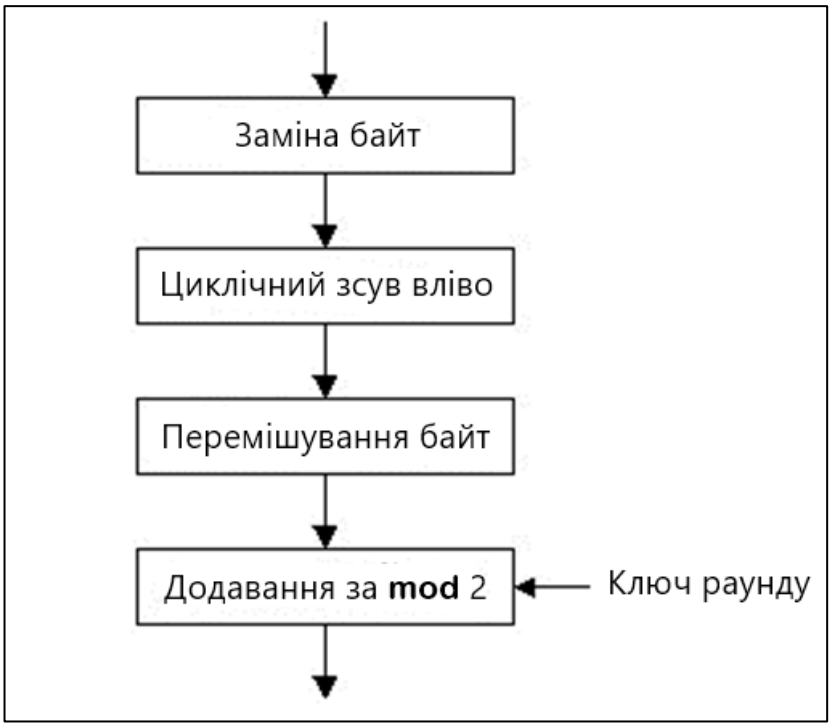

Рисунок 1.9 – Схема раунду AES.

Останній раунд відрізняється від усіх інших тим, що в ньому не виконується крок «Перемішування байт».

Процес розшифровки шифротексту AES аналогічний процесу шифрування у зворотному порядку. Кожен раунд складається з чотирьох процесів, проведених у зворотному порядку:

- Додавання за mod 2
- Перемішування байт
- Циклічний зсув вліво
- Заміна байт

Оскільки кроки в кожному раунді перебувають у зворотному порядку, на відміну від шифру Фейстеля, алгоритми шифрування та дешифрування повинні бути окремо впроваджені, хоча вони дуже тісно пов'язані [11].

Раундові ключі отримуються із ключа шифру *K* за допомогою функції розширення ключа. В результаті формується масив раундових ключів, із якого потім безпосередньо вибирається необхідний раундовий ключ.

Кожен раундовий ключ має довжину 128 біт або чотири 4-байтних слова $w_i, w_{i+1}, w_{i+2}, w_{i+3}$. Перші чотири слова w_0, w_1, w_2, w_3 в масиві ключів містять ключ шифра, а із інших 40 отриманих слів вибираються по 4 слова для ключу раунду. Слова вибираються наступним чином: перші чотири слова є ключом під номером 0, наступні чотири слова w_4, w_5, w_6, w_7 – раундовим ключом для першого раунду і так далі.

Нові слова $w_{i+4}, w_{i+5}, w_{i+6}, w_{i+7}$ наступного раундового ключа визначаються із слів $w_i, w_{i+1}, w_{i+2}, w_{i+3}$ попереднього ключа на основі рівнянь:

$$w_{i+5} = w_{i+4} \oplus w_{i+1};$$
$$w_{i+6} = w_{i+5} \oplus w_{i+2};$$
$$w_{i+7} = w_{i+6} \oplus w_{i+3}.$$

Перше слово w_{i+4} в кожному раунді змінюється по іншому:

$$w_{i+4} = w_i \oplus g(w_{i+3}).$$

Робота функції g складається з 3 кроків:

1. циклічний зсув 4-байтного слова вліво на один байт;

2. заміна кожного байту слова, отриманого в кроці 1, згідно з таблицею SubBytes, що використовується при шифруванні;

3. додавання по *mod* 2 байтів, отриманих в кроці 2, з раундовою константою $RC[i] = (RC[i], 0,0,0)$, несекретною і унікальною для кожного раундового ключа K_i. Три праві байти цієї константи – нульові, а ненульовий лівий байт змінюється наступним чином:

$$RC[1] = 1, RC[i] = 2 \cdot RC[i-1], i = 1,2, \dots ,10 \text{ [12]}.$$

Ціль додавання з раундовими константами – знищити будь яку симетрію, що може з'явитися на різних етапах розгортання ключа і привести до появи слабких ключів, як в алгоритмі DES.

Переваги алгоритму AES:

1. Широке застосування. Після того, як уряд США затвердив його, як стандарт, він підтримується більшістю постачальників.

2. Вибір з трьох можливих довжин ключів, дозволяє користувачам знайти компроміс між швидкістю та безпекою. Відповідно, збільшення довжини ключа збільшує час виконання шифрування та розшифрування. В той же час, всі три розміри ключа вважаються безпечними і найбільш відомі атаки на AES зменшують ефективний розмір ключа в найкращому раз на 3 біти.

3. AES використовує лише один S-box для всіх байтів у всіх раундах. DES, в свою чергу, використовує вісім різних S-box, які збільшують вимоги до реалізації.

Недоліки AES:

1. Використання дуже простої алгебраїчної структури.

2. Кожен блок зашифрований однаковим чином.

3. Складна програмна реалізація.

В 2006р. Державна служба спеціального зв'язку та інформаційної безпеки України оголосила Український національний криптографічний конкурс, головною метою якого було вибір симетричного блочного алгоритму шифрування, що став би новим національним стандартом України замість ГОСТ 28147-89. В результати конкурсу серед усіх представлених алгоритмів було обрано шифр «Калина», що був встановлений як ДСТУ 7326:2014.

«Калина» - це алгоритм, що базується на AES та підтримує розміри блок і ключів до 512 біт. Кількість раундів залежить від розміру блоку і дорівнює 10, 12, 14 при розмірах блоку 128, 256 та 512 відповідно. Розмір ключа може дорівнювати або бути більшим за розмір блоку.

Процес шифрування за допомогою алгоритму «Калина» може бути описаний за допомогою наступної формули:

$$T_{l,k}^{(K)} = \eta_l^{(K_t)} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \prod_{v=1}^{t-1} (\kappa_l^{(K_v)} \circ \psi_l \circ \tau_l \circ \pi_l') \circ \eta_l^{(K_0)}$$

Де K – ключ шифрування розміром k біт,

$\eta_l^{(K_v)}$ – функція додавання раундового ключа до матриці станів за модулем 2^{64},

$\acute{\pi}_l$ – заміна байтів у матриці станів,

τ_l – перестановка елементів матриці станів (циклічний зсув байт вправо),

ψ_l – лінійне перетворення елементів матриці стану над скінченим полем,

$\kappa_l^{(K_v)}$ – додавання за *mod* 2 матриці стану і раундового ключа K_v.

Більш наглядна схему роботи алгоритму шифрування «Калина» зображено на рис. 1.9.

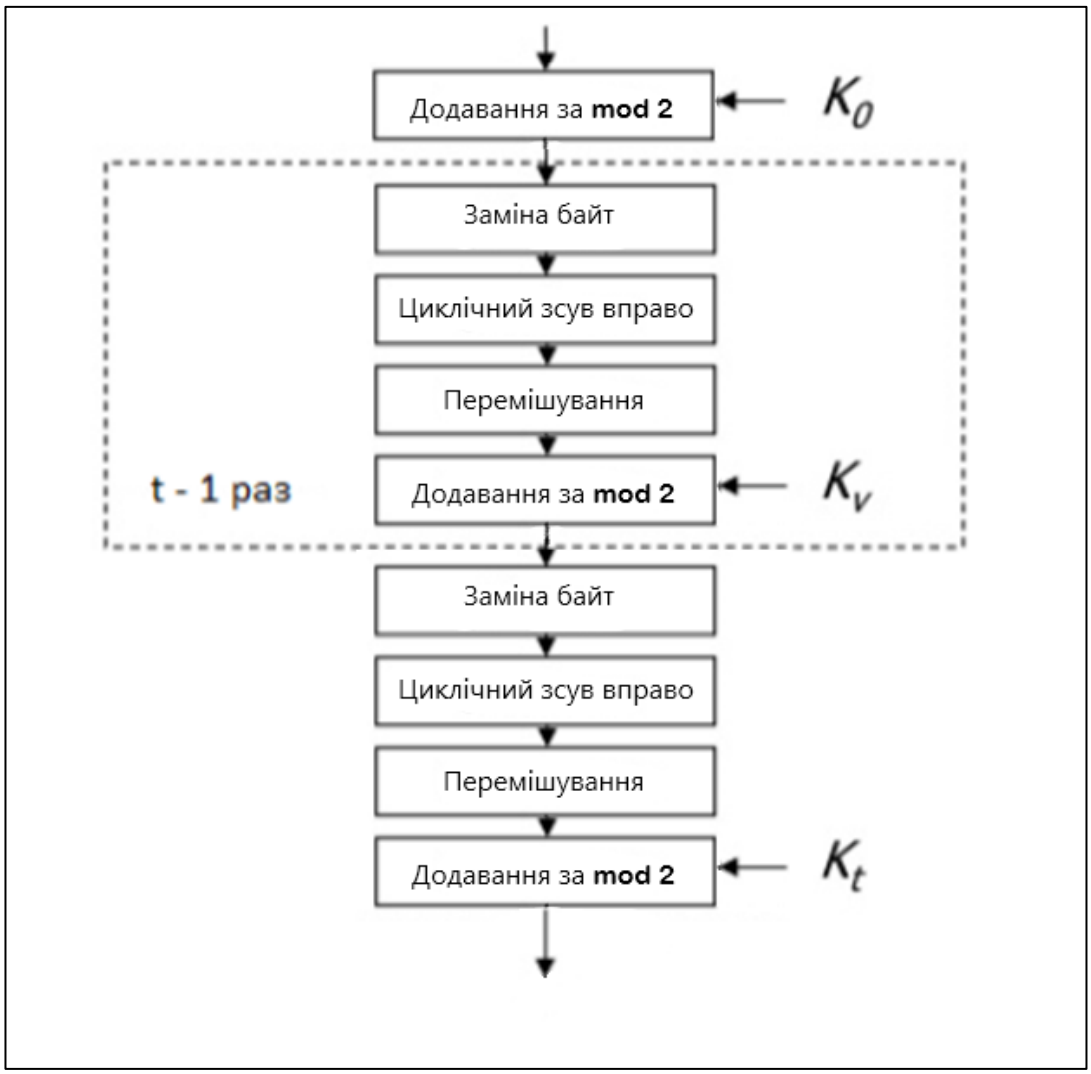

Рисунок 1.10 – Схема роботи шифру «Калина».

Генерація раундових ключів відбувається у два етапи:

1. Визначення проміжного ключа з ключа шифрування;
2. Визначення раундових ключів з проміжного ключа.

Розмір проміжного ключа K_σ дорівнює розміру блока і представлений у вигляді матриці розмірами $8 \times c$. Даний ключ генерується з ключа шифрування K за допомогою наступного перетворення:

$$\theta^{(K)} = \psi_l \circ \tau_l \circ \pi_l' \circ \eta_l^{(K_\alpha)} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \eta_l^{(K_\omega)} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \eta_l^{(K_\alpha)}$$

Якщо розмір блока і ключа шифру однакові тоді $K_\alpha = K_\omega = K$. Якщо ж розмір ключа в два рази більший за розмір блока, то $K_\alpha \parallel K_\omega = K$.

Кожен раундовий ключ представлений у вигляді матриці розмірами $8 \times c$. Генерація раундових ключів залежить від ключу шифру, проміжного ключу та індексу раунду.

Раундові ключі K_i з парними індексами ($i \in \{0,2, \dots, t\}$) обчислюються за допомогою наступного перетворення:

$$\Xi^{(K,K_\sigma,i)} = \eta_l^{(\varphi_i^{(K_\sigma)})} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \kappa_l^{(\varphi_i^{(K_\sigma)})} \circ \psi_l \circ \tau_l \circ \pi_l' \circ \eta_l^{(\varphi_i^{(K_\sigma)})}$$

Де $\varphi_i^{(K_\sigma)} = \eta_l^{(K_\sigma)}(\vartheta \ll (\frac{i}{2}))$, де $\vartheta = \mu_l^{(0x00010001\dots0001)}$.

Якщо розмір ключа дорівнює розміру блока, то $K \ggg 32 \cdot i$ – вхідний аргумент для перетворення $\Xi^{(K,K_\sigma,i)}$.

Якщо розмір ключа не дорівнює розміру блока, то як вхідні аргументи для перетворення $\Xi^{(K,K_\sigma,i)}$ використовуються:

$L_{k,l}(K \ggg 16 \cdot i)$, для генерації раундових ключів з парними індексами, що діляться на 4 ($i \in \{0,4,8, \dots\}$);

$R_{k,l}(K \ggg 64 \cdot [\frac{i}{4}])$, для генерації раундових ключів з парними індексами, що не діляться діляться на 4 ($i \in \{2,6,10, \dots\}$).

Раундові ключі з непарними індексами обчислюються за допомогою раундового ключа минулого раунду з використанням формули:

$$K_i = K_{i-1} \lll \left(\frac{l}{4} + 24\right) \text{[13].}$$

1.2.2 Асиметричні алгоритми шифрування

Прикладом такого типу алгоритмів є RSA. Робота цього алгоритму складається з 4 етапів: генерації ключів, шифрування, розшифрування та передача ключів.

Генерація ключів відбувається за наступною схемою:

1. Беруться два великі прості числа p і q;
2. Вираховується добуток цих чисел $m=p \cdot q$;
3. Знаходиться функція Ейлера $\varphi(n) = (p-1)(q-1)$;
4. Вибирається ціле число e таке, що $1 < e < \varphi(n)$ та e взаємно просте з $\varphi(n)$;
5. Знаходиться таке число d, що $ed \equiv 1 (\text{mod } \varphi(n))$.

Відповідно для того, щоб одна сторона могла надсилати секретні повідомлення іншій, необхідно, щоб друга сторона, передала свій відкритий ключ *(n, e)*.

Для того, щоб зашифрувати якесь повідомлення *M*, необхідно спочатку перетворити M в ціле число m так, щоб $0 \leq m < n$. Після цього обчислити зашифрований текст, за допомогою рівняння:

$$c = m^e \ mod \ n$$

Для того, щоб розшифрувати повідомлення, отримувачу потрібно обчислити наступну рівність [14]:

$$m = c^e \ mod \ n$$

Головним недоліком асиметричних алгоритмів шифрування є те, що вони на 3-4 порядки повільніші за симетричні , тому для того щоб здійснювати шифрування та розшифрування необхідно мати великі обчислювальні можливості, що є дуже великою проблемою. Саме тому їх зазвичай використовують у комбінації з симетричними алгоритмами.

1.3 Алгоритм шифрування на базі підстановок довільної розрядності

В роботі [15] запропоновано алгоритм шифрування на базі найпростіших регулярних лінійних структурах. Регулярними структурами називаються ОККМ, що базуються на однакових однорозрядних конструктивних модулях (КМ), які можуть реалізувати до 850 різних підстановок з довільною розрядністю. Таким чином, програмна і апаратна реалізація таких структур є доволі простою.

Нехай є множина підстановок $P = \{p_1(X), p_2(X), \dots, p_k(X)\}$ однакової розрядності, де $X =< x_{n-1}, \dots, x_1, x_0, >$ кортеж двійкових значень, n – розрядність. Тоді для двох будь яких значень x_1 та x_2 із X, і при цьому $x_1 \neq x_2$ вірно наступне: $p_i(x_1) \neq p_i(x_2)$. А також для кожної підстановки $p_i(X) \in P$ існує обернена підстановка $q_i(X) \in Q$, така що $q_i(p_i(X)) = p_i(q_i(X)) = X$.

Суперпозицією підстановок із P вважається наступна підстановка $P_m(X)$:

$$P_m(X) = p_{i1}\left(p_{i2}\left(\dots\left(p_{im}(X)\right)\dots\right)\right).$$

В такому разі розшифрування відбувається за допомогою використання наступної суперпозиції:

$$q_{i1}\left(q_{i2}\left(\dots\left(q_{im}(X)\right)\dots\right)\right),$$

де q_{ij} – це обернені підстановки p_{ij}, $i = \{1,2,\dots,|P|\}, j = \{1,2,\dots,m\}$. В такому разі секретним ключем буде послідовний перелік номерів підстановок із P.

Також оскільки розміри сучасних файлів завжди кратні 8 бітам, то для даного алгоритму було реалізовано множину M з 8-розрядними підстановками та фіксованими значеннями h та r. Та визначено множини P та Q, де |P| = |Q| = 192.

Таким чином перед виконанням алгоритму потрібно сформувати таблиці вихідних значень Y для всіх можливих вхідних значень X та будь яких h, r ∈ {0,1}. Для 8-розрядного ОККМ X ∈ {0,1,…,255}.

Після цього відбувається формування ключа. Вибір заданої кількості m підстановок із множини P відбувається випадковим чином, для цього використовується криптостійкий генератор псевдовипадкових чисел. Тоді загальна кількість ключів становить $|P|^m$. Кожен ключ являє собою m-розрядне число в позиційній системі числення з основою 192. Таким чином розмір ключа знаходиться в межах від 7,4×m до 7,6×m. В такому разі при m=17 розрядність ключів в даному алгоритмі буде рівна розрядності ключів в алгоритмі AES, а при m=34 – розрядності ключів ГОСТ 28147-89.

Для кожної підстановки визначаються значення на бокових виходах згідно з визначеним раніше порядком. Значення на виході визначається за допомогою сформованих раніше таблиць, відповідно до значень на бокових входах та вхідного значення X.

Отже, до відкритих даних алгоритму відносяться: перелік підстановок із визначених раніше 192, перелік булевих функцій на значень h, r.

До закритих даних відносяться: ключ з m випадково вибраних підстановок.

Алгоритм розшифрування відрізняється від алгоритму шифрування лише тим, що послідовність підстановок в ключі використовується в оберненому порядку до шифрування [15].

1.4 Порівняння алгоритмів шифрування

Перевагами симетричного шифрування є:

1. Простота. Даний тип шифрування легко здійснити, оскільки необхідно всього лиш вказати та розповсюдити секретний ключ, а потім за його допомоги виконувати шифрування та розшифрування повідомлення.

2. Швидкість. Повідомлення шифрується дуже швидко порівняно з асиметричними алгоритмами шифрування.

3. Запобігає поширенню загрози безпеки повідомлення. Для кожної групи людей використовується різний секретний ключ. Якщо ключ скомпрометовано, то це відобразиться лише на певній парі відправника та отримувача. Зв'язок з іншими людьми залишиться безпечним.

Недоліками ж є:

1. Якщо противник, що перехоплює повідомлення, визначить ключ, то він зможе миттєво дешифрувати все, що було зашифровано з допомогою цього ключа. А також, зможе створювати фальшиві повідомлення вводячи в оману отримувача.

2. Передавання ключів. Ключі повинні передаватись з надзвичайною безпекою, оскільки вони можуть надати доступ до інформації, що зашифрована з їх допомогою. Особливо це є проблемою коли необхідно часто змінювати ключі.

3. Кількість ключів збільшується квадратично до кількості людей, що обмінюються інформацією. Таким чином кількість ключів, що необхідні одному користувачу можна описати наступним виразом:

$$N = \frac{n \cdot (n-1)}{/2}$$

де N – кількість ключів, n – кількість людей, що обмінюються інформацією. Тоді при $n = 3$, $N = 3$, а при $n = 4$, $N = 6$ і при $n = 10$, $N = 45$.

4. Симетричні ключі є предметом силових атак (методом повного перебору), коли всі ключі з множини ключів систематично намагаються зламати шифрування.

«Але постійне підвищення швидкості передачі даних по каналах зв'язку ставить проблему забезпечення реального часу криптографічних перетворень. Існуючі стандартні алгоритми шифрування далеко не завжди

вирішують цю проблему в сучасних умовах, тим менш ймовірно зможуть в майбутньому» [16].

Більшість відомих алгоритмів шифрування використовують такі операції як: пересилання даних, порозрядне додавання за модулем 2 (XOR), додавання, віднімання та циклічний зсув. Нехай, для оцінки швидкості алгоритмів шифрування, ці операції виконуються з однаковою швидкістю, яка буде рівна одиниці. Тоді швидкість алгоритму на підстановка можна визначити як $2wm$, де w – кількість байт відкритого тексту, що шифрується, m – кількість підстановок в ключі.

В алгоритмі ГОСТ 28147-89 блоки мають розмір 8 байт, для шифрування використовуються 32 цикли Фейстеля. В кожному такому циклі виконуються операція додавання, 8 операцій пересилання даних, операція циклічного зсуву, операції XOR та пересилання для завершення циклю – всього виходить 12 операцій. Таким чином для шифрування одного такого блоку виконується $32 \times 12 = 384$ операції, або 48 операцій для одного байту.

В алгоритмі AES шифруються блоки по 16 байт, за 10 раундів. В кожному раунді виконуються такі операції – XOR, 16 операцій пересилання, 3 операції циклічного зсуву та операція множення на матрицю, що потребує 64×4 операції пересилання та 16×3 операції XOR. Таким чином, загальна кількість операцій для 10 раундів, з урахуванням того, що в останньому раунді не виконується множення на матрицю, дорівнює 1200 операцій, або ж 75 операцій на байт.

В результаті стає зрозуміло, що шифрування з використанням підстановок може мати перевагу в швидкості, лише при умові, що кількість підстановок в ключі не буде більшою ніж 24, що дозволяє можливість спрогнозувати хорошу криптостійкість при великій швидкості [17].

Можливим напрямком вирішення проблеми є використання підстановок довільної розрядності, які програмно або апаратно реалізуються

регулярними структурами лінійної складності від кількості розрядів. Алгоритм який базується на таких регулярних структурах було описано в підрозділі 1.3.

В [18] на основі експериментів показано, що швидкість шифрування, з використанням таких структур, на декілька порядків перевищує швидкість шифрування стандартних алгоритмів (табл. 1.1).

«В роботах [19, 20] теоретично доведено, що прямі та обернені підстановки можуть реалізуватись на ОККМ з будь якою розрядністю $m \geq 1$ КМ, при умові, що всі КМ, наприклад, належать типу 2а. Таким чином постає необхідність у вдосконаленні алгоритму шифрування на базі підстановок довільної розрядності шляхом використання багато розрядних КМ» [16].

Таблиця 1.1 - Показники швидкодії реалізацій алгоритмів Калина, AES при розмірі ключа 128 біт та алгоритму шифрування, який грунтується на суперпозиції підстановок із найпростіших регулярних ОККМ.

Розмір файлу	**Калина, 10 раундів**	**AES, 10 раундів**	**ОККМ**
10 байт	0,297 с	0,093 с / 0,062 с	0,000000165 с
100 байт	1,266 с	0,640 с / 0,500 с	0,000000896 с
1000 байт	10,828 с	6,000 с / 3,860 с	0,000008197 с
10000 байт	106,563 с	59,594 с / 38,484 с	0,00008109 с
100000байт	1099,844 с	612,235 с / 396,343 с	0,000808815 с

Висновки до розділу 1

Отже, було проаналізовано існуючі стандарти шифрування, а саме їх схему роботи, генерацію ключів та швидкість. Також розглянуто алгоритм шифрування на базі підстановок довільної розрядності. Та проведено порівняльну характеристику швидкості шифрування зі стандартами. З

результатів порівняння, можна дійти до висновку, що алгоритм шифрування на базі регулярних структур є на порядок швидшим, за рахунок того, що для шифрування одного байту інформації він використовує на порядок менше операції в порівнянні зі стандартами. Особливо це помітно при шифруванні великих об’ємів даних.

Таким чином, постає необхідність у подальшому дослідженні алгоритму на базі регулярних структур та можлива його модифікація.

РОЗДІЛ 2
ПОДАЛЬШИЙ РОЗВИТОК АЛГОРИТМІВ КРИПТОГРАФІЧНИХ ПЕРЕТВОРЕНЬ НА ПІДСТАНОВКАХ ДОВІЛЬНОЇ РОЗРЯДНОСТІ

2.1 Одновимірні каскади конструктивних модулів

В сучасній комп'ютерній інженерії один із традиційних методів прискорення перетворення інформації – реалізація їх на комбінаційних схемах, наприклад на базі програмованих логічних інтегральних схем (ПЛІС).

ПЛІС – це електронний компонент, що застосовується для створення цифрових інтегральних схем. На відміну від цифрових мікросхем, логіка роботи ПЛІС не визначається при виготовленні, а задається за допомогою програмування. Для програмування використовуються програматори та відлагоджувальні середовища, що дозволяють задати необхідну структуру цифрового пристрою у вигляді принципової електричної схеми або програми на спеціальних мовах опису апаратури, таких як: Verilog, VHDL, AHDL [21].

Існує кілька типів ПЛІС, що зараз використовуються:

- Complex Programmable Logic Device (CPLD) – складний програмований логічний пристрій. Він містить в собі великі програмовані логічні блоки – макрокомірки, що з'єднані з зовнішніми виходами та внутрішніми шинами.
- Field-Programmable Gate Array (FPGA) – містить блоки множення-додавання, що широко застосовуються при обробці сигналів, а також логічні елементи і їх блоки комутації. FPGA зазвичай використовується для обробки сигналів, мають більше логічних елементів та більш гнучку архітектуру ніж CPLD.

Підстановка необмеженої розрядності в ПЛІС відбувається за допомогою використання ОККМ – рис. 2.1.

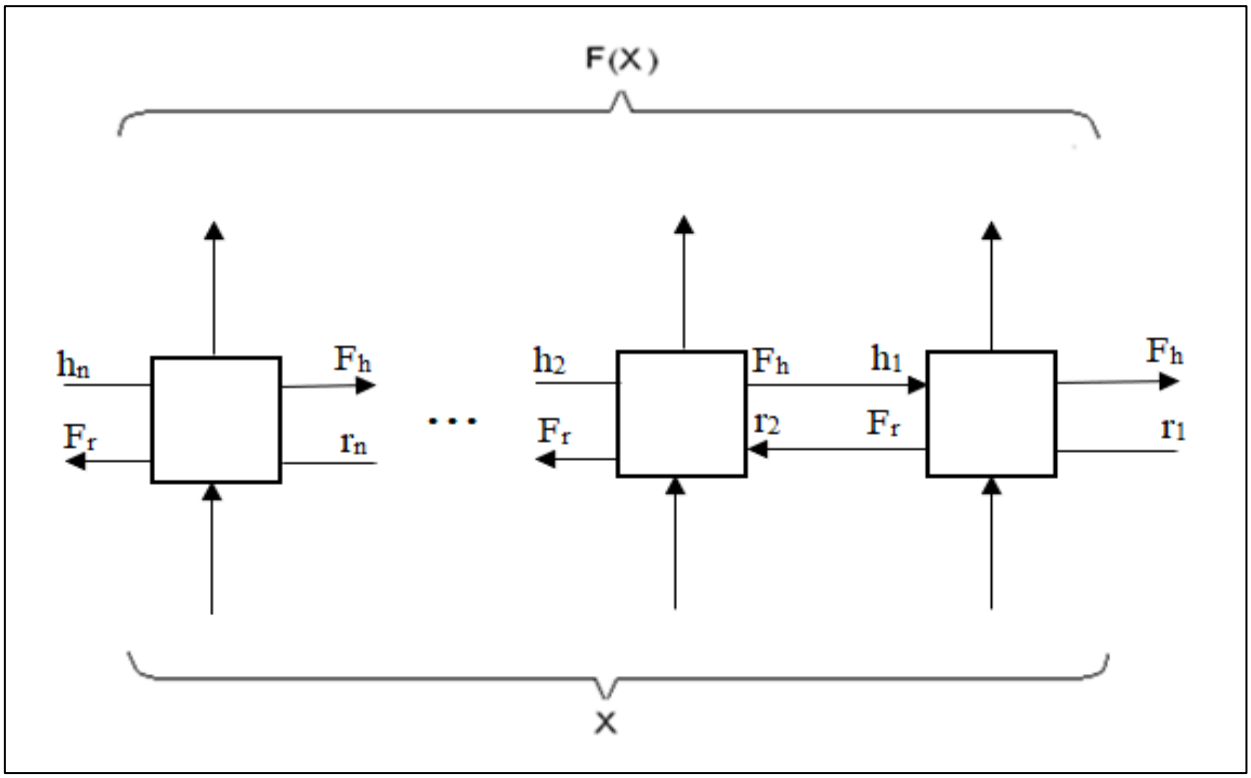

Рисунок 2.1 – Одновимірний каскад конструктивних модулів.

ОККМ складаються з послідовно з'єднаних КМ. Структура КМ зображена на рис. 2.2. Необхідно зазначити, що в ОККМ всі КМ є однаковими.

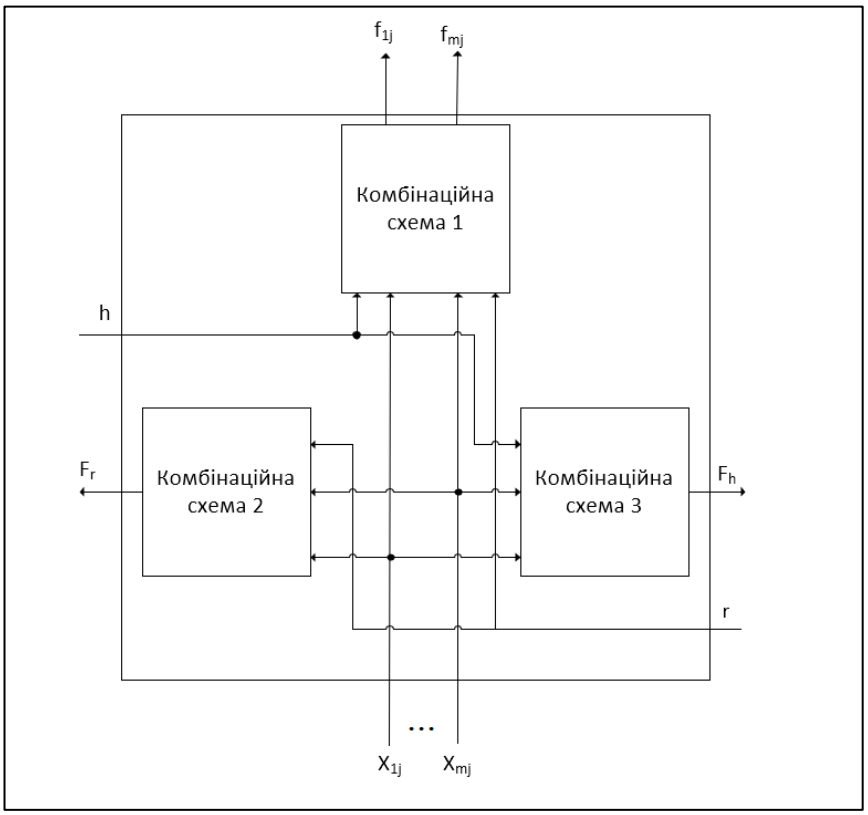

Рисунок 2.2 – Структура конструктивного модуля.

Як видно на рис. 2.2. кожен КМ складається з трьох комбінаційних схем (КС), роботу яких можна описати наступними булевими функціями:

1. Функція $f_h(h, x_1..x_m)$ на правому боковому виході (КС 2).
2. Функція $f_r(r, x_1..x_m)$ на правому боковому виході (КС 3).
3. m – функцій $f_i(h, x_1..x_m, r)$ $(i = 1,2,..,m)$ на первинному виході КМ (КС 1).

Відповідно на вхід КМ поступають значення $(x_{1j}..x_{mj})$, які передаються на вхід всіх трьох комбінаційних схем. Таким чином вихід КС2 поступає на вхід КС2 та вхід КС1 конструктивного модулю, що знаходиться ліворуч від поточного, а КС3 – на вхід КС3 та вхід КС1 конструктивного модулю, що знаходиться праворуч від поточного. Отже, в залежності від аргументів, що поступають на вхід КМ, та вихідних значень КС2 і КС3 формується вихід КС1, що є підстановкою для вхідний аргументів.

2.2 Апаратна реалізація алгоритму шифрування на базі лінійних структур

Для апаратної реалізації булевих функцій КМ, можна використати так звані Look-Up Table (LUT).

LUT являє собою матрицю істинності, яка дозволяє визначити логіку поведінки, в нашому випадку КМ. Тобто, це таблиця яка визначає, що буде на виході, для будь якого значення на вході.

FPGA зазвичай реалізує комбінаторну логіку, за допомогою таких LUT, і коли FPGA виконує налаштування, то при цьому виконується заповнення таблиці вихідних значень, що називається “LUT-Mask”, та складається з бітів SRAM. Після цього в залежності від поєднання вхідних змінних, наданих користувачем, відповідний біт пам'яті з'явиться на виході LUT. Це пов'язано з тим, що вхідні біти, надані користувачем, діють як лінії вибору для мультиплексорів, присутніх в LUT.

Таблиці підстановок в LUT можна записувати за допомогою біт-потоку, який дозволяє завантажити дані для конфігурації FPGA. В такому разі LUT можна буде переналаштувати у будь який момент часу завантаживши туди нові таблиці підстановок.

Таким чином, для того, щоб реалізувати КМ з 2-розрядними підстановками, можна використати дві LUT, що містить 5 входів та 2 виходи. Але необхідними є тільки 4 входи. В такому випадку на 5-ий вхід необхідно буде подавати, якесь константне значення. І оскільки відповідно 5-ий вхід не відіграє ніякої ролі у формуванні виходу КМ, то було вирішено не зображати його на схемі КМ.

LUT, що реалізуються булеві функції КМ, мають схеми, що зображені на рис. 2.3 та на рис. 2.4.

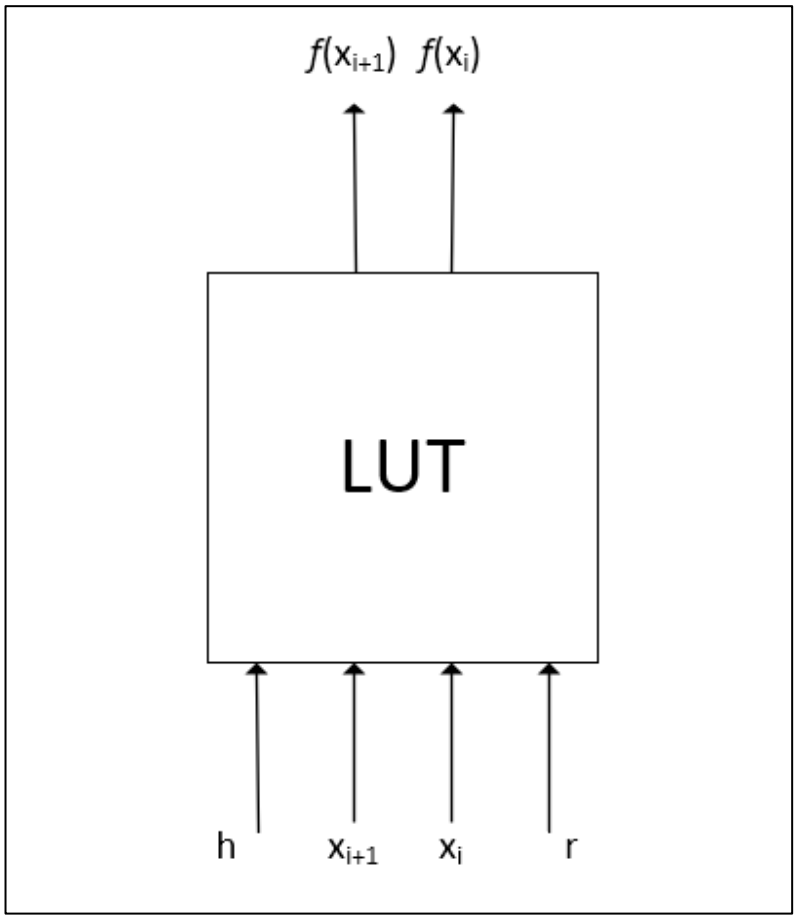

Рисунок 2.3 – Схема LUT, що реалізує булеву функцію підстановки для 2-розрядного КМ.

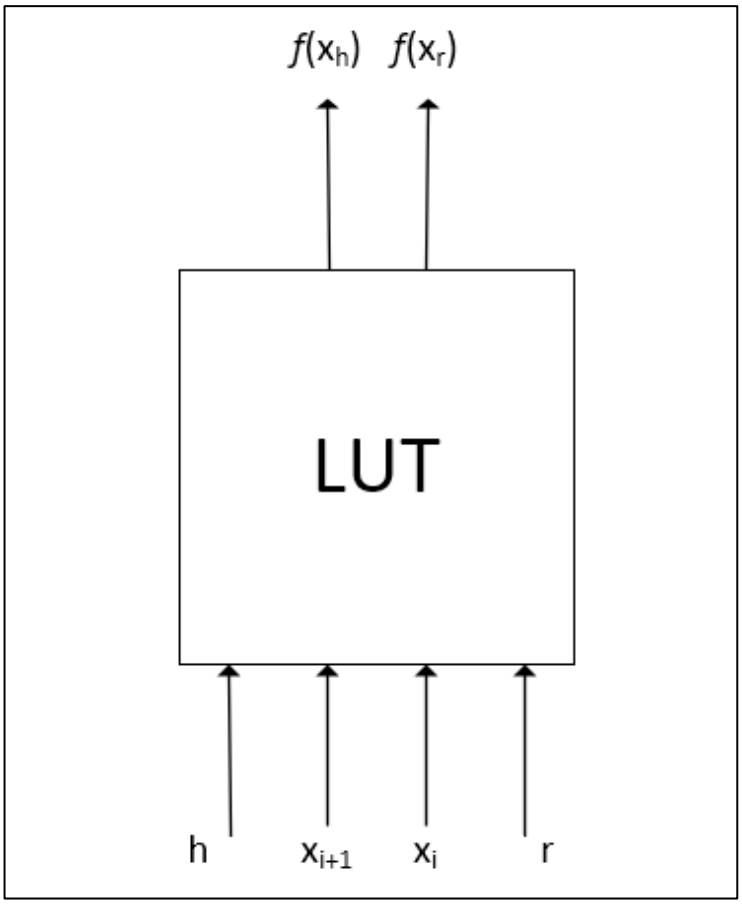

Рисунок 2.4 – Схема LUT, що реалізує булеві функції бокових виходів 2-розрядного КМ.

Як видно на рис. 2.3 та 2.4 LUT мають, абсолютно однакові входи, різниця лише в таблицях значень, що завантажуються в ці LUT. Також при побудові КМ, на базі цих LUT, необхідно забезпечити подачу вхідних значень КМ на входи LUT. Таким чином, схема КМ має вигляд, що зображений на рис. 2.5.

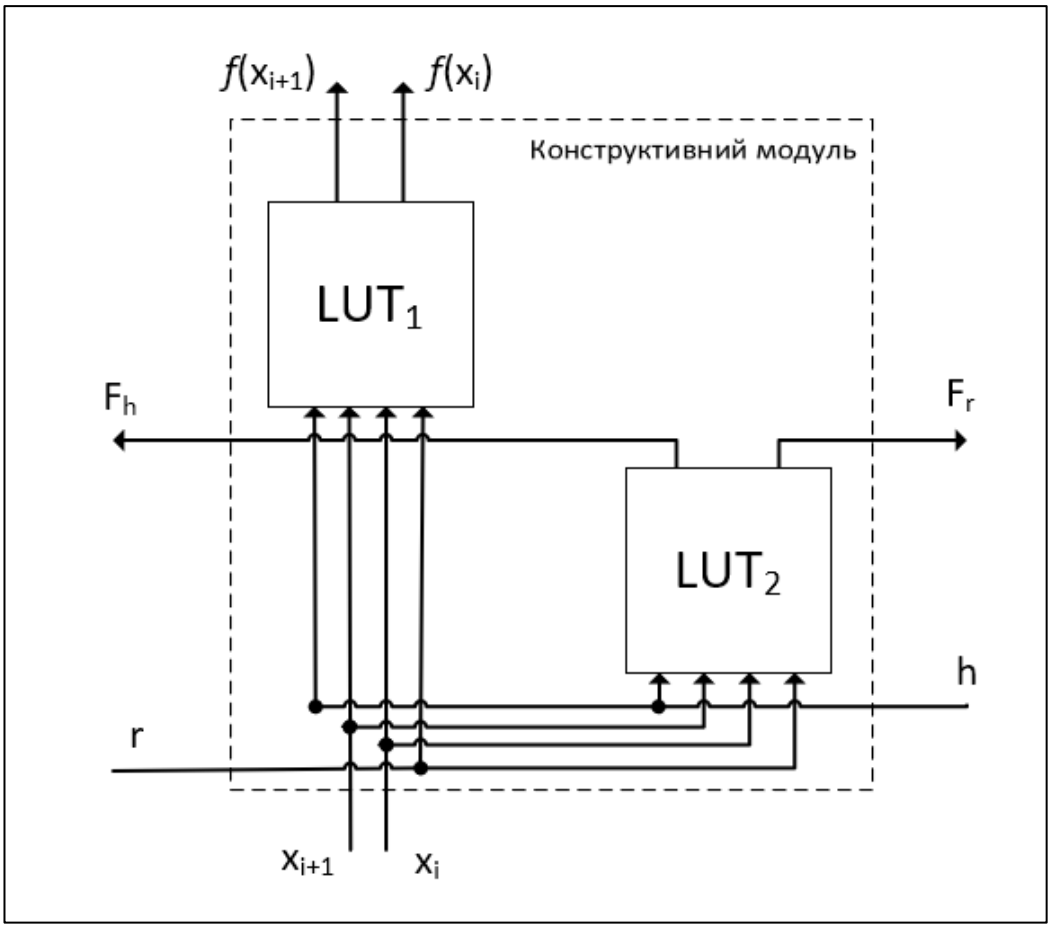

Рисунок 2.5 – Схема 2-розрядного КМ

В такому разі ОККМ, що базується на такому КМ буде мати наступний вигляд (рис. 2.6):

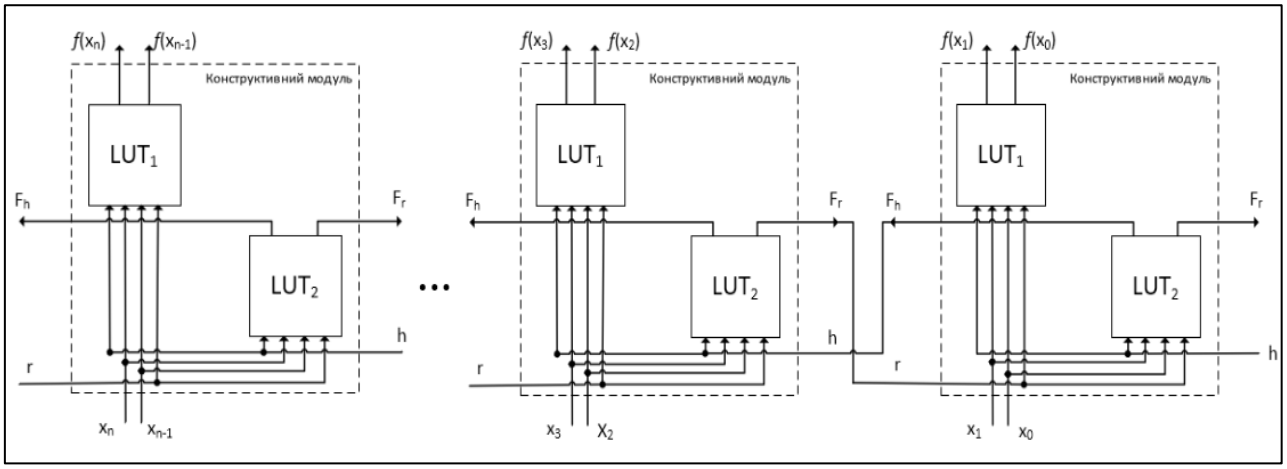

Рисунок 2.6 – Схема ОККМ на базі 2-розрядного КМ.

Для того, щоб реалізувати 4-розрядний КМ, необхідно використати LUT, що має 6 входів та 1 вихід. Тоді схема LUT, що реалізує булеву функцію підстановки має наступний вигляд (рис. 2.7):

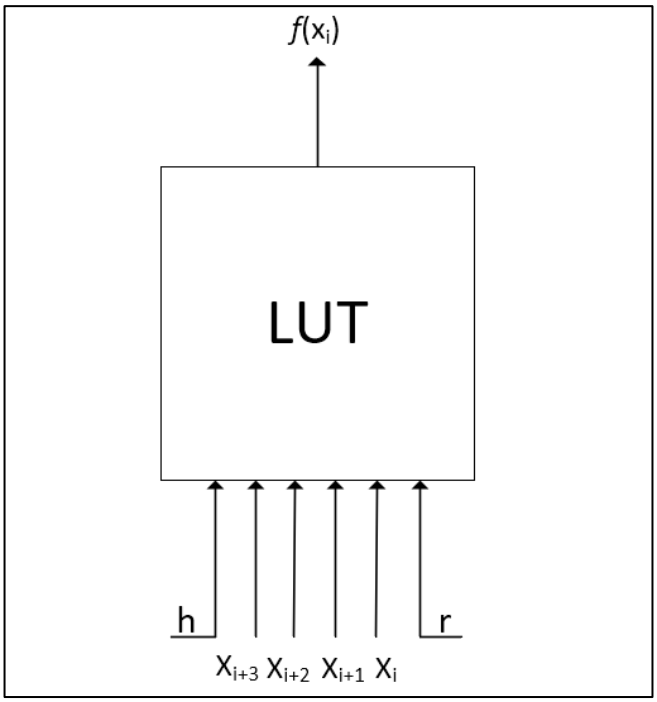

Рисунок 2.7 – Схема LUT, що реалізує булеву функцію підстановки.

Оскільки, дана схема LUT, має лише один вихід, то відповідно вона може здійснити лише підстановку для одного розряду. Тоді для того, щоб здійснити підстановку з 4 розрядів, потрібно послідовно з’єднати такі LUT (рис. 2.8).

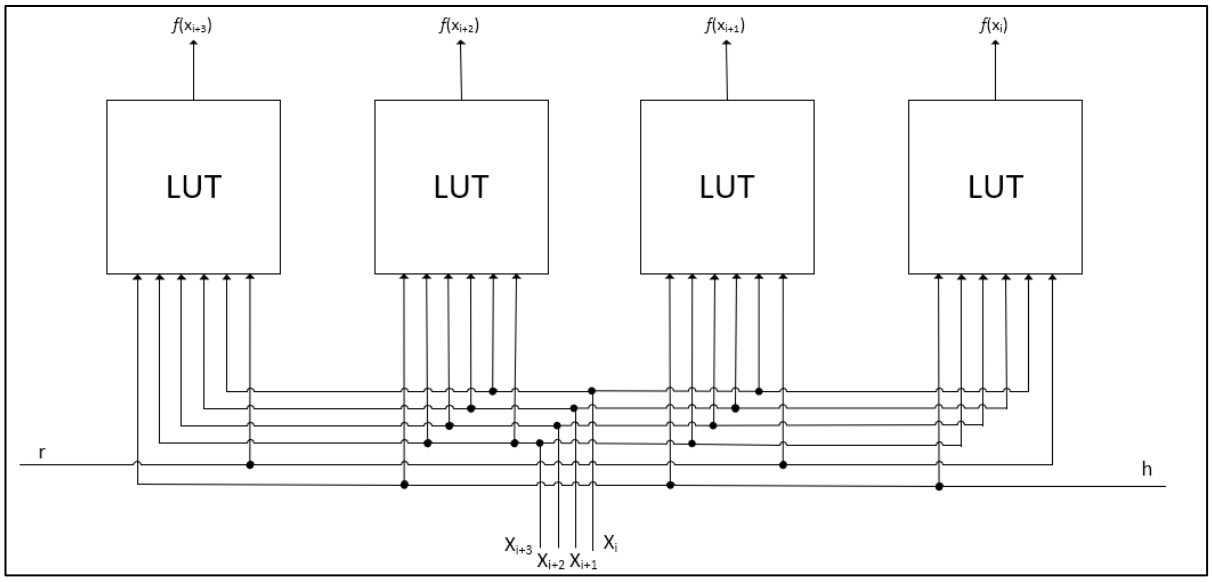

Рисунок 2.8 – Схема реалізації 4-розрядних підстановок на базі LUT.

Для реалізації бокових булевих функцій, необхідно використати таку саму схему LUT, як і на рис. 2.7, але на її виході буде одна із тих функцій

(рис 2.9). Так як, на формування значення бокової функції f_h значення r не потрібне і навпаки для - f_r не потрібне значення h, то відповідні входи немає сенсу зображати на схемі.

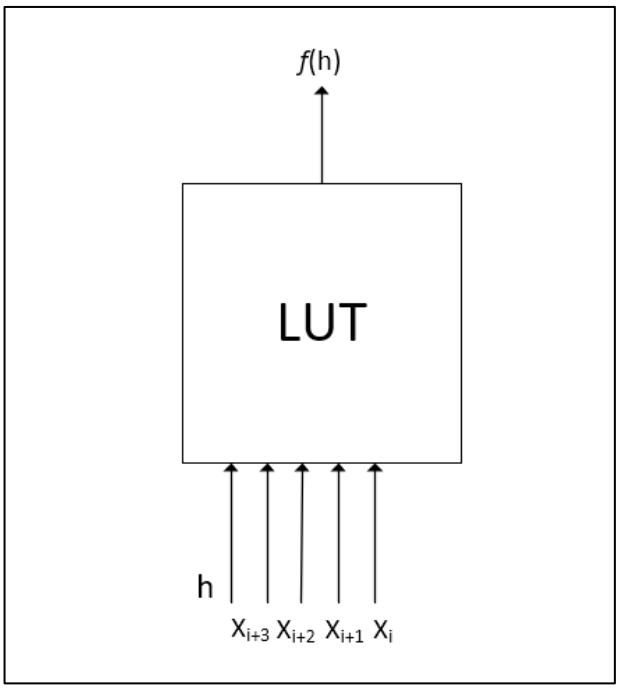

Рисунок 2.9 – Схема LUT для бокової функції f_h

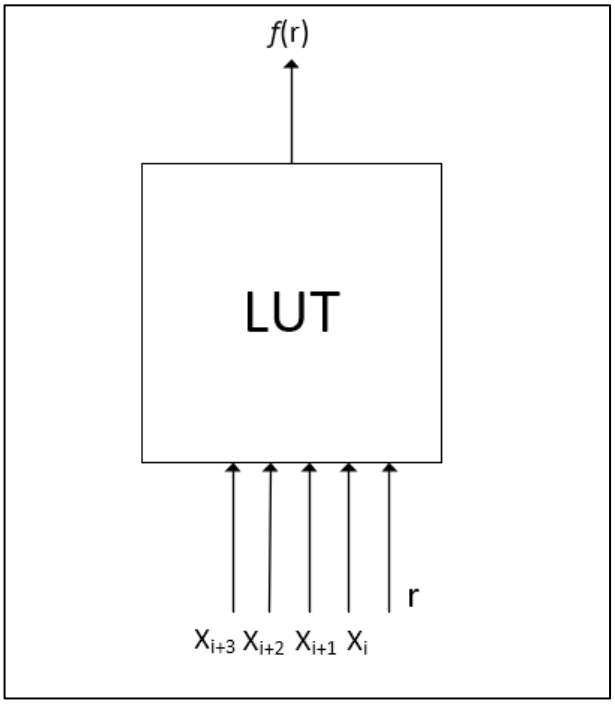

Рисунок 2.10 – Схема LUT для бокової функції f_r.

Отже, для того, щоб реалізувати КМ з 4-розрядними підстановками, необхідно скомбінувати схеми, що зображені на рис. 2.8 - 2.10. В результаті було утворену схему КМ, що зображена на рис. 2.11.

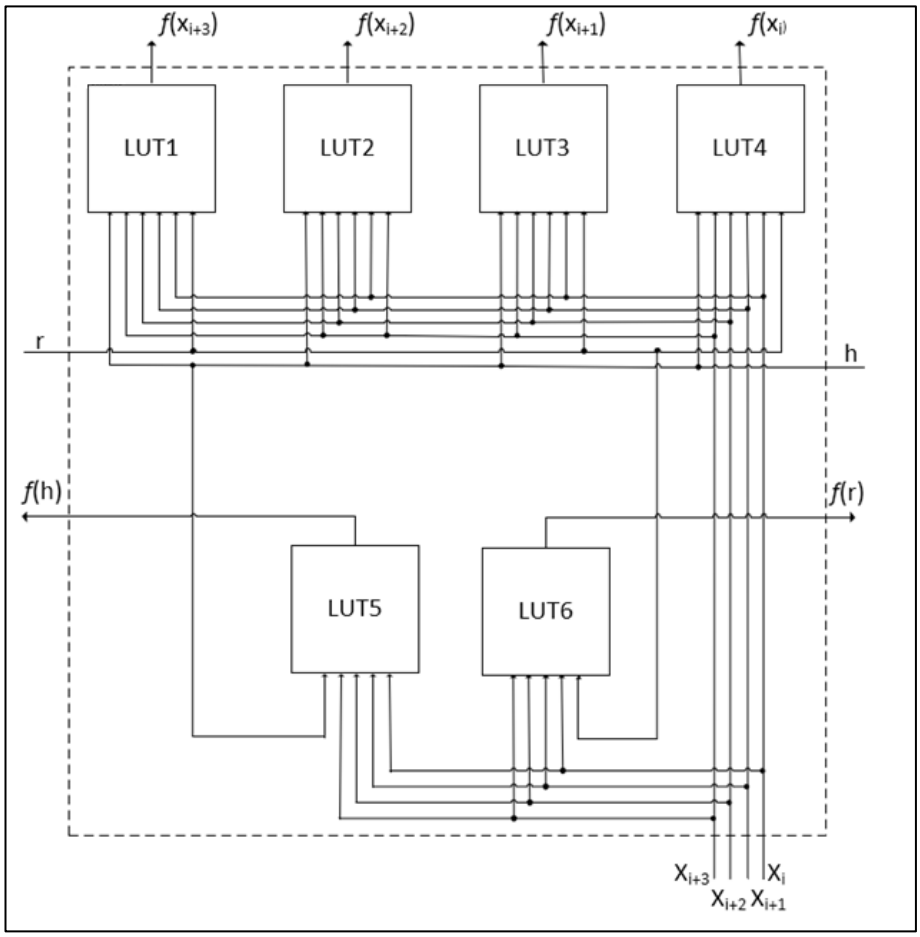

Рисунок 2.11 – Схема 4-розрядного КМ на базі LUT.

В такому разі, для формування ОККМ на базі 4-розрядних КМ, необхідно послідовно з’єднати такі КМ наступним чином:(рис. 2.12).

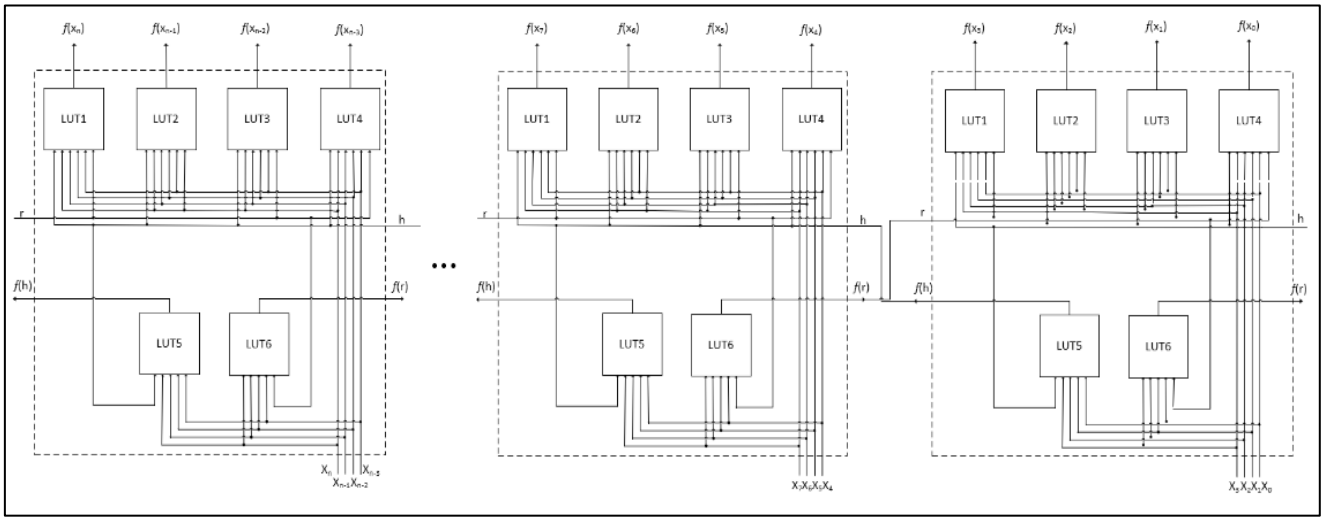

Рисунок 2.12 – Схема ОККМ на базі 4-розрядних КМ.

В результаті час затримки на виконання підстановки можна визначити, за допомогою, наступної формули:

$$(S-1)\cdot t_{LUT} + t_{LUT} = S \cdot t_{LUT}$$

Де S – кількість КМ в ОККМ, а t_{LUT} – час, який потрібен LUT, для визначення значення значень на виході.

Дана формула має такий вигляд, оскільки перед виконанням функції підстановки, необхідно спочатку визначити значення на бокових входах КМ.

Якщо порівнювати зі стандартними алгоритмами симетричного шифрування, наприклад з AES, то йому для шифрування 1 байту інформації необхідно виконати 75 операцій, як було визначено в підрозділі 1.4. Припустимо, що для виконання однієї операції необхідно 1нсек, тоді AES зашифрує 1 байт за 75 нсек. В свою чергу апаратна реалізація ОККМ на базі 4-розрядних КМ виконає шифрування за:

$$\frac{S \cdot t_{LUT}}{\frac{S}{2}} = \frac{S \cdot 1\text{нсек}}{\frac{S}{2}} = 2\text{нсек.}$$

Де S – кількість КМ в ОККМ, $\frac{S}{2}$ – кількість байт, які можна зашифрувати за допомогою такого ОККМ, оскільки для шифрування 1 байту необхідно здійснити дві 4-розрядні підстановки.

Отже, як видно з розрахунків, даний алгоритм є на порядок швидшим за сучасні стандарти шифрування.

Також можна здійснити апаратну реалізацію даного алгоритму з використанням суперпозиції підстановок. В такому випадку схема для 2-розрядних підстановок буде наступною (рис. 2.13):

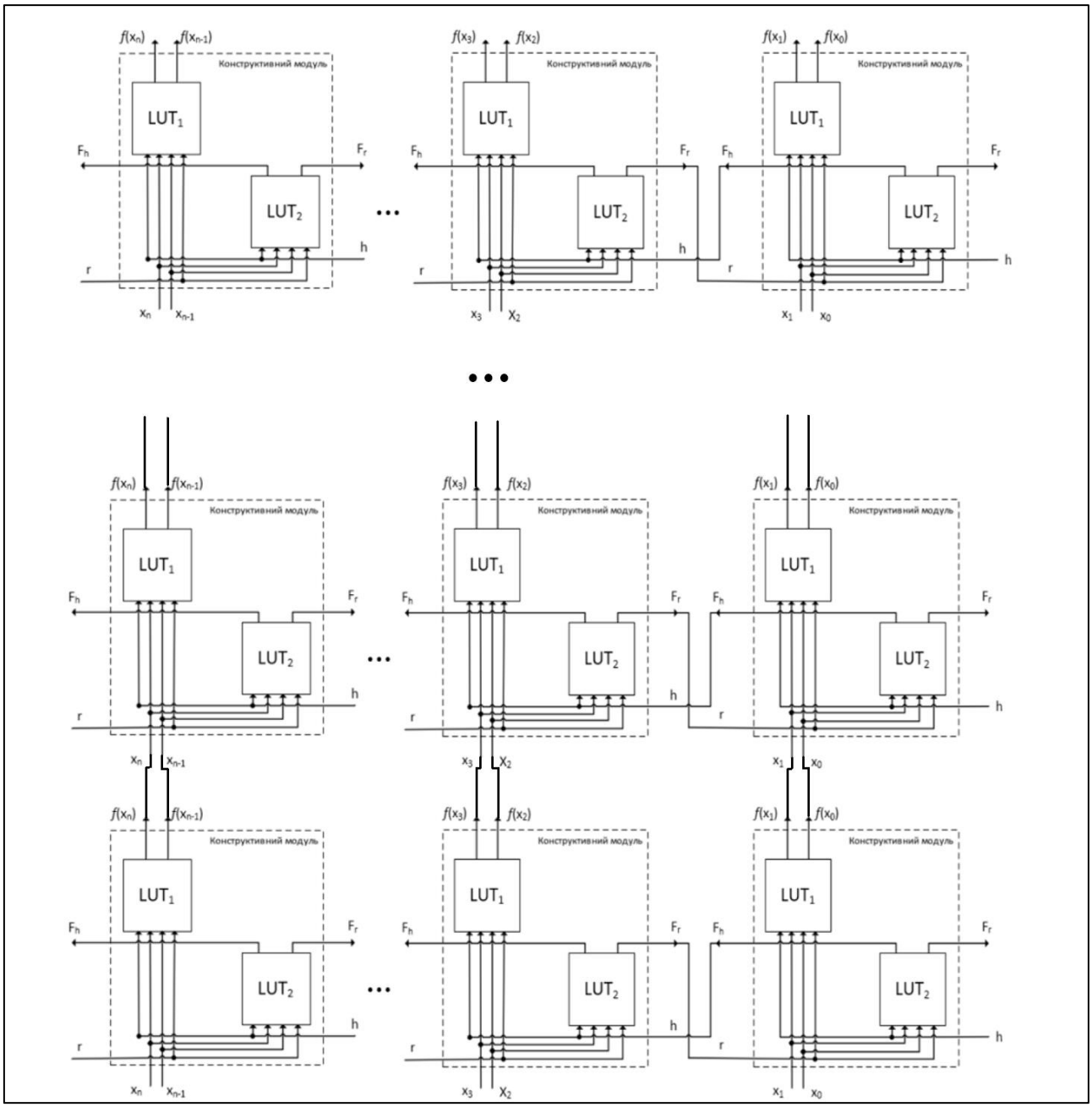

Рисунок 2.13 – Схема апаратної реалізації алгоритму шифрування на базі лінійних структур з використанням суперпозиції підстановок для 2-розрядних підстановок

В такому разі для 4-розрядни підстановок схема апаратної реалізації алгоритму зображена на рис. 2.14.

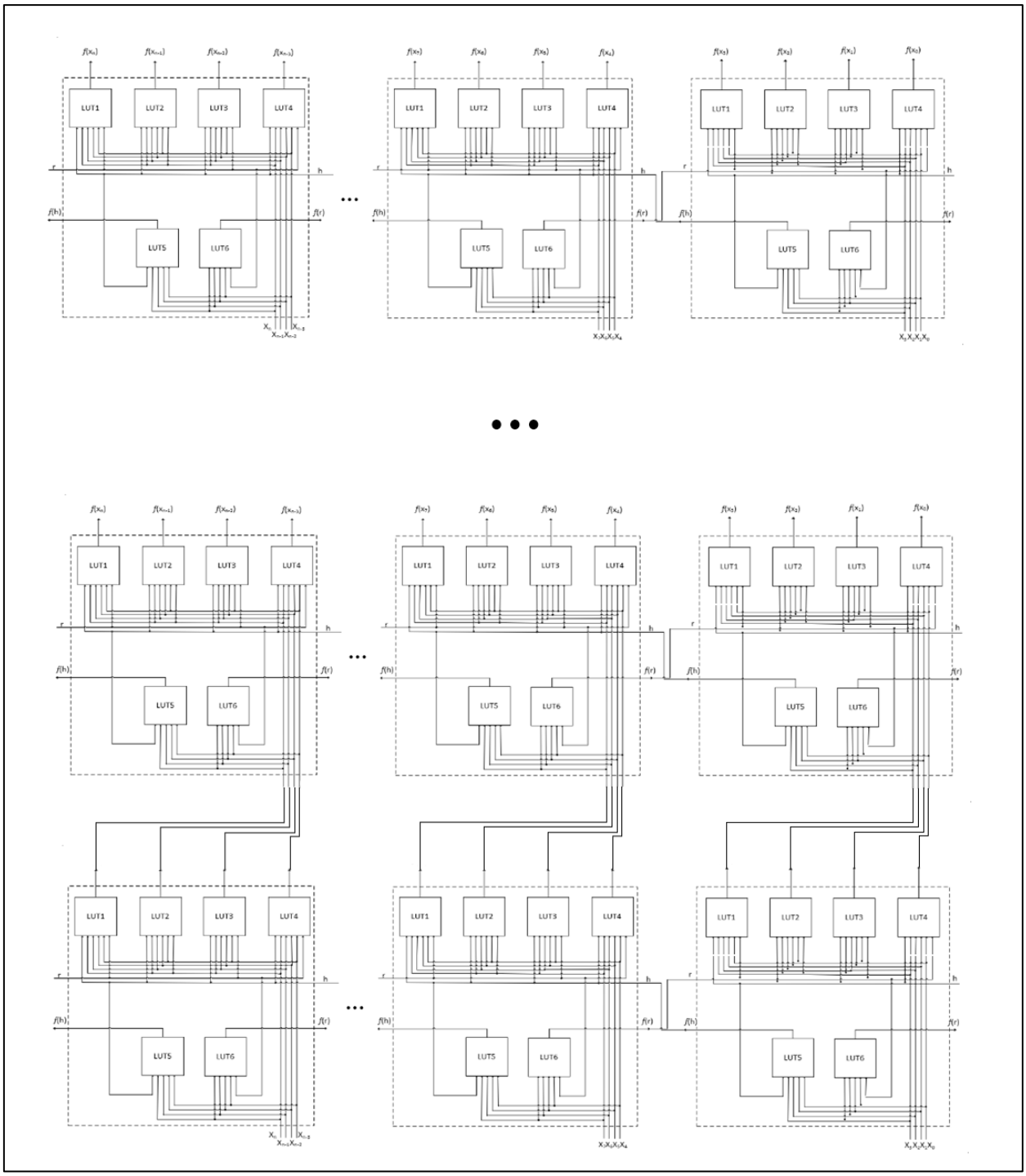

Рисунок 2.14 – Схема апаратної реалізації алгоритму шифрування на базі лінійних структур з використанням суперпозиції підстановок для 4-розрядних підстановок

2.3 Програмна реалізація алгоритму шифрування на базі лінійних структур

Функції $f_i(h, x_1..x_m, r)$ $(i = 1,2..m)$ на первинному виході КМ розглядаються як кортеж (впорядкований набір) $F(h, x_1..x_m, r) =< f_1(h, x_1..x_m, r) ... f_m(h, x_1..x_m, r)$. Таким чином кожен із кортежів $F(0, x_1..x_m, 0)$, $F(0, x_1..x_m, 1)$, $F(1, x_1..x_m, 0)$, $F(1, x_1..x_m, 1)$ реалізує підстановку – одну із $(2^m)!$

При програмній реалізації підстановки представлені у вигляді масивів на 2^m елементів. Кожен такий елемент містить не менш ніж m-розрядне двійкове число. Позначимо $F(0, x_1..x_m, 0)$ як M_{00}, аналогічно для $F(0, x_1..x_m, 1)$ як M_{01}, $F(1, x_1..x_m, 0)$ як M_{10}, $F(1, x_1..x_m, 1)$ як M_{11}. Таким чином можна задати $((2^m)!)^4$ різних кортежів $F(h, x_1..x_m, r)$. Програмно масиви M_{00}, M_{01}, M_{10}, M_{11} можна задати у вигляді одного масиву $M[h, X, r]$ розмірністю $M[0..1, 0..2^m - 1, 0..1]$. Отже, масиви підстановок задаються випадковим чином, їх елементи є різними та приймають значення із множини $\{0, ..., 2^m - 1\}$, так само як і вхідний аргумент X.

В свою чергу бокові функції $f_h(h, x_1..x_m)$ та $f_r(r, x_1..x_m)$ приймають значення із множини $\{0, 1\}$. При програмній реалізації дані бокові функції представлені у вигляді двох масивів вибору підстановок $M_h[h, X]$ та $M_r[r, X]$ з розмірністю $[0..1, 0..2^m - 1]$.

Для того, щоб забезпечити реалізацію підстановок будь якої розрядності масиви вибору підстановок створюються на основі масивів підстановок:

- масив $M_h[0, 0..2^m - 1]$ на основі масивів M_{00} та M_{01};
- масив $M_h[1, 0..2^m - 1]$ на основі масивів M_{10} та M_{11};
- масив $M_r[0, 0..2^m - 1]$ на основі масивів M_{00} та M_{10};
- масив $M_r[1, 0..2^m - 1]$ на основі масивів M_{01} та M_{11}.

Спочатку масиви вибору підстановок вважаються заповненими нулями і значення в них генеруються за допомогою наступного алгоритму:

1) випадковим чином визначається індекс *i* пустого елемента масиву вибору підстановок, та даному елементу присвоюється значення 1;

2) дістається значення елементу з індексом *i* з другого масиву відповідної пари масивів підстановок;

3) елемент з таким значенням шукається в першому масиві відповідної пари масивів підстановок;

4) якщо елемент масиву вибору підстановок із індексом знайденого елементу пустий, то йому присвоюється значення 1 та п.п.2-4 повторюються знову, інакше виконання алгоритму завершується.

Програмна реалізація даного алгоритму наступна:

```
kmr1:=(1 shl m)-1; // kmr1=2^m-1
  for j:=0 to kmr1 do fbok[j]:=0;
  src:=random(kmr1+1);
  next:=fbok[src];
  while (next=0) do
  begin
    fbok[src]:=1;
    raby:=funy[src];
    for i:=0 to kmr1 do
    begin
      if raby=funx[i] then
      begin
        src:=i;
        next:=fbok[i];
        goto lbn;
      end;
    end;
```

```
  ShowMessage('error');
 lbn:
 end;
```

Де fbok – будь який із масивів $M_h[0], M_h[1], M_r[0], M_r[1]$, funx, funy – відповідна пара масивів M.

Для виконання процедури шифрування необхідно задати значення h0 та r0 для крайніх КМ. В алгоритмах шифрування та розшифрування використовуються лише операції пересилання. Теоретично, завдяки наявності таблиць вибору, всі біти результату шифрування залежать від всіх бітів початкового повідомлення.

Алгоритм шифрування наступний:

1. вхідне повідомлення розбивається на масив елементів заданої розрядності;
2. задаються початкові значення h та r;
3. для допоміжного масиву задається лічильник *i* з початковим значенням 0;
4. r присвоюється значення з масиву вибору підстановок Mr[r, file[i]], де file – масив вхідного повідомлення;
5. *i*-му елементу допоміжного масиву присвоюється нове значення r;
6. лічильник *i* збільшується на 1;
7. якщо значення лічильника дорівнює розміру масиву вхідного повідомлення, то виконання продовжується з п. 8, інакше починаючи з п. 4;
8. лічильнику *i* присвоюється значення, що дорівнює довжині масиву вхідного повідомлення мінус 1;
9. *i*-му елементу допоміжного масиву присвоюється присвоюється значення з масиву підстановок M[h, [file[i], temp[i]], де temp – допоміжний масив, file – масив вхідного повідомлення;
10. h присвоюється значення з масиву вибору підстановок Mh[h, file[i]], де file – масив вхідного повідомлення;

11. значення лічильника *i* зменшується на 1;

12. якщо значення лічильника дорівнює -1, виконання алгоритму припиняється, інакше продовжується починаючи з п. 9. В результаті зашифроване повідомлення знаходиться в допоміжному масиві.

Програмна реалізація алгоритму шифрування виглядає наступним чином:

```
For i:=1 to W do
Begin
        Temp[i]:= r0;
        r0:=Mr[r0,file[i]];
End;
For i:= W  downto 1 do
Begin
        Temp[i]:=M[h0, file[i], Temp[i]];
        H0:= Mh[h0,file[i]]
End;
```

Де file – масив байт вхідного файлу, що необхідно зашифрувати, temp – допоміжний масив для результуючого файлу такого ж розміру, cm – об’єкт з даними конструктивного модулю, такими як масив підстановок та масиви вибору підстановок, які можна отримати за допомогою методів getM(), getMh(), getMr() відповідно. Також необхідно задати початкові значення h0 та r0.

2.4 Реалізація розшифровування

Для шифрування та розшифрування довільних файлів використовується один і той же алгоритм, змінюються лише таблиці підстановок та таблиці вибору підстановок. Для розшифрування повідомлення зашифрованого за допомогою даного алгоритму використовуються обернені КМ. Масиви MO_{00}, MO_{01}, MO_{10}, MO_{11} є

масивами підстановок оберненого КМ. Програмно їх можна задати у вигляді одного масиву $MO[h, X, r]$. Вони формуються із масивів M_{00}, M_{01}, M_{10}, M_{11}, таким чином щоб $MO[h, M[h, X, r], r]$ дорівнювало X, при будь якому $0 \leq X \leq 2^m - 1$. Також формування масивів $MO[h, X, r]$ доцільно сумістити з формуванням масивів $MO_h[h, X]$ та $MO_r[r, X]$ в наступному порядку:

- вхідні дані - M_{00}та $M_h[0, X]$, вихідні - M_{00} та $MO_h[0, X]$;
- вхідні дані - M_{01}та $M_r[0, X]$, вихідні - M_{01} та $MO_r[0, X]$;
- вхідні дані - M_{10}та $M_r[1, X]$, вихідні - M_{10} та $MO_r[1, X]$;
- вхідні дані - M_{11}та $M_h[1, X]$, вихідні - M_{11} та $MO_h[1, X]$.

Алгоритм формування масивів оберненого КМ наступний:

1. задається лічильник *i* з початковим значенням 0;

2. значення *i*-го елементу відповідного масиву підстановок M записується в допоміжну змінну;

3. елементу масиву підстановок оберненого КМ індекс якого дорівнює значенню в допоміжній змінній присвоюється значення лічильника *i*;

4. елементу масиву вибору підстановок оберненого КМ індекс якого дорівнює значенню в допоміжній змінній присвоюється значення *i*-го елементу відповідного масиву вибору підстановок;

5. значення лічильника збільшується на 1;

6. якщо значення лічильника дорівнює розміру масиву підстановок, то виконання алгоритму завершується, інакше продовжується з п. 2.

Програмна реалізація алгоритму формування масивів оберненого КМ наступна:

```
kmr1:=(1 shl m)-1; // kmr1=2^m-1
for i:=0 to kmr1 do
  begin
    temp:=fosrc[i];
    fodst[temp]:=i;
    fbdst[temp]:=fbsrc[i];
```

```
end;
```

Де Fosrc – вхідний масив M_{xy} прямого КМ, Fodst – вихідний масив M_{xy} оберненого КМ, Fbsrc – вхідний масив для моделювання відповідної бокової функції прямого КМ, Fbdst– вихідний масив для моделювання відповідної бокової функції оберненого КМ.

Як було зазначено вище, алгоритм розшифрування аналогічний алгоритму шифрування, оскільки для розшифрування використовуються обернені КМ, тому програмна реалізація алгоритму наступна:

```
For i:=1 to W do
Begin
        Temp[i]:= r0;
        r0:=MOr[r0,file[i]];
End;
For i:= W  downto 1 do
Begin
        Temp[i]:=MO[h0, file[i], Temp[i]];
        H0:= MOh[h0,file[i]]
End;
```

2.5 Модифікований алгоритм формування таблиць вибору підстановок

З метою збільшення кількості різних таблиць вибору підстановок було запропоновано модифікацію до вже існуючого алгоритму генерації таких таблиць. «Початкові значення в даних масивах вважаються пустими, і в подальшому генеруються на основі пар масивів підстановок за наступним алгоритмом (при початковому значенні d=2):

1) визначається індекс i чергового пустого елемента масиву вибору підстановок, та даному елементу присвоюється значення d, якщо пусті елементи відсутні, то алгоритм закінчує свою роботу;

2) дістається значення елементу з індексом *i* з другого масиву відповідної пари масивів підстановок;

3) елемент з таким значенням шукається в першому масиві відповідної пари масивів підстановок;

4) якщо елемент масиву вибору підстановок із індексом знайденого елементу пустий, то йому присвоюється значення *d* та п.п.2-4 повторюються знову, інакше виконується п.5;

5) значення *d* збільшується на 1, перехід на п1.

В результаті роботи алгоритму, множина індексів масиву вибору підстановок розбивається на *d*-1 класи. Особливістю індексів одного і того класу полягають в тому, що співпадають множини значень масивів підстановок при цих індексах. В масиві вибору підстановок будь який із класів позначається 0 або 1. Це дає можливість сформувати 2^{d-1} різних таблиць вибору підстановок, що може використовуватись як додатковий елемент при генерації ключа шифруваннях» [16].

Програмна реалізація даного модифікованого алгоритму на мові Java наступна:

```
Arrays.fill(fbok, -1);
int d = 0;
int i;
while ((i = getNextElementIndex(fbok)) != -1) {
   do {
      fbok[i] = d;
      int src = funy[i];
      for (int j = 0; j < funx.length; j++) {
         if (funx[j] == src) {
            i = j;
            break;
         }
```

```
        }
    } while (fbok[i] == -1);
    d++;
}
```

Де fbok – будь який із масивів $M_h[0], M_h[1], M_r[0], M_r[1]$, funx, funy – відповідна пара масивів M.

Метод getNextElementIndex() повертає значення індекс наступного пустого елемента масиву вибору підстановок та має наступну реалізацію:

```
int getNextElementIndex(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == -1) {
            return i;
        }}
    return -1;
}
```

Нехай, наприклад, є дві таблиці з 4-розрядними підстановками табл. 2.1 та табл. 2.2.

Таблиця 2.1 – Перша таблиця підстановок з відповідної пари

Індекс	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Значення	12	8	5	0	11	10	3	9	1	14	2	7	4	13	6	15

Таблиця 2.2 – Друга таблиця підстановок з відповідної пари

Індекс	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Значення	1	10	4	12	0	8	2	3	6	15	7	9	14	5	11	13

В такому випадку в результаті роботи алгоритму буде утворено таблицю вибору підстановок зі значеннями в табл. 2.3.

Таблиця 2.3 – Згенерована таблиця вибору підстановок

Індекс	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Значення	2	3	4	2	2	3	5	5	2	4	5	5	4	4	2	4

Таким чином, в даному прикладі було згенеровано таблицю вибору підстановок, що складається з 4 класів. Після цього значення даних класів можна замінити на значення з множини $\{0, 1\}$ в результаті чого може бути утворено 2^4 різних таблиць вибору підстановок.

2.6 Програмна реалізація алгоритму шифрування на базі лінійних структур з використанням суперпозиції підстановок

З метою кращого захисту від атак зі спеціально підібраними початковими повідомленнями або атак зі спеціально підібраними криптограмами запропоновано модифікацію до оригінального алгоритму шифрування на базі лінійних структур.

Модифікація полягає в тому, що генерується множина КМ з **Q** елементів, яка не є секретною. В такому разі шифрування та розшифрування відбувається за допомогою підстановки, яка є суперпозицією підстановок із випадково вибраних КМ (операція множення в групі підстановок) (рис. 2.3) [22].

В такому разі генерація масивів підстановок та масивів вибору підстановок в КМ відбувається аналогічно оригінальному алгоритму. Таким чином створюється множина публічних КМ.

Алгоритм шифрування ж має певні відмінності та виглядає наступним чином:

1. задається значення кількості КМ, що будуть використовуватися при шифруванні;
2. вхідним аргументом є повідомлення, що потрібно зашифрувати;
3. задається лічильник зі значенням рівним 0;
4. випадковим чином обирається КМ із множини публічних КМ;
5. відбувається шифрування згідно оригінального алгоритму, на вхід йому передається вхідний аргумент;

6. вхідному аргументу присвоюється значення виходу оригінального алгоритму;

7. значення лічильника збільшується на 1;

8. якщо значення лічильника дорівнює заданій кількості КМ, що використовуються при шифруванні, то виконання алгоритму завершується, інакше алгоритм продовжує виконуватися починаючи з п. 4.

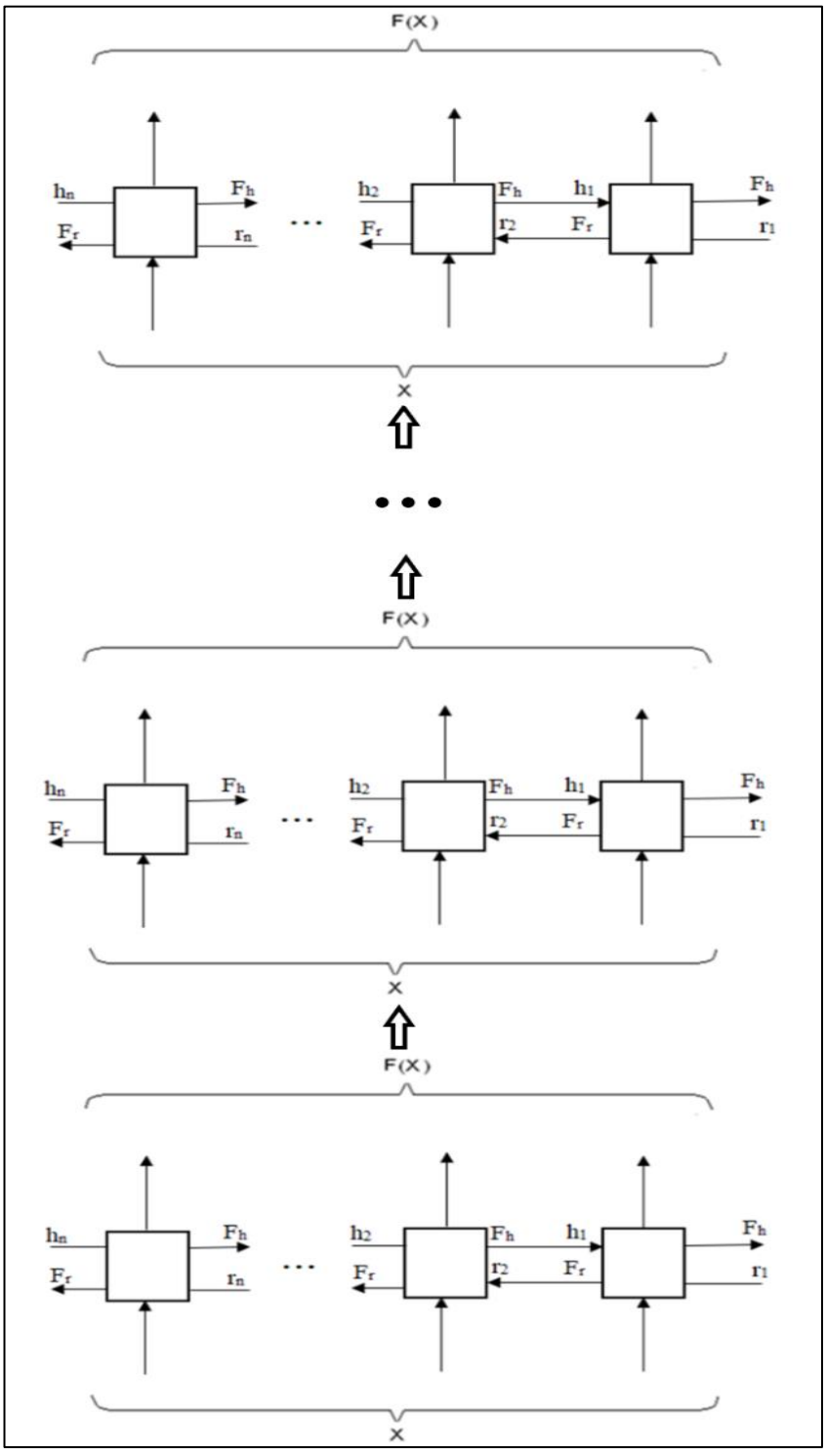

Рисунок 2.3 Схема алгоритму шифрування на базі лінійних структур з використанням суперпозиції підстановок.

Програмна реалізація даного алгоритму наступна:

```
int[] encodedData = data;
CMData cm;
for (int i = 0; i < m; i++) {
   cm = cmPool[substitutionOrder[i]];
   for (int j = 0; j < data.length; j++) {
      encodedData[j] = r0;
      r0 = cm.getMr()[r0][data[j]];
   }
   for (int j = data.length - 1; j >= 0; j--) {
      encodedData[j] = cm.getM()[h0][encodedData[j]][data[j]];
      h0 = cm.getMh()[h0][data[j]];
   }
}
```

Де data – вхідний масив з даними, що треба зашифрувати, encodedData – допоміжний масив який зберігає в собі вхідні значення до кожного ОККМ, cm – об’єкт в якому містяться масиви підстановок та масиви вибору підстановок кожного конкретного КМ, cmPool – множина згенерованих КМ.

В результаті зашифрований текст вхідного повідомлення, після виконання даного алгоритму, міститься в змінній encodedData.

Але для того, щоб даний алгоритм шифрування мав сенс, необхідно впевнитись чи не являє алгоритм шифрування на базі лінійних структур собою групу, де елементами будуть підстановки, а операціями між елементами – суперпозиція підстановок. Оскільки, якщо алгоритм є групою, то немає сенсу здійснювати суперпозицію підстановок, так як вихід може бути еквівалентним виходу підстановки, що базується на одному ОККМ. Наприклад, доведено, що алгоритм DES, не являється групою, саме тому

стала можливою реалізація алгоритму 3DES. Тому було створено програмну систему, з метою перевірки чи являється цей алгоритм групою.

Розроблена програмна система має наступну структуру (рис 2.4):

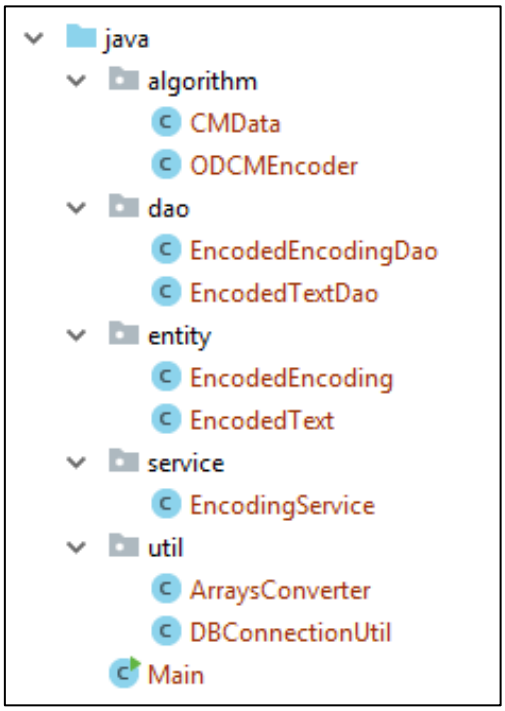

Рисунок 2.4 – Структура програмної системи.

В класі CMData містяться наступні поля:

private final int[][][] M;

private final int[][] Mr;

private final int[][] Mh;

private final int[][][] MO;

private final int[][] MOr;

private final int[][] MOh;

private final int m;

Де масив M – це масив усіх підстановок, Mr та Mh – відповідні масиви вибору підстановок, MO – обернений масив підстановок, MOr та MOh – відповідні обернені масиви вибору підстановок, m – розрядність підстановок.

Усі ці масиви ініціалізуються в конструкторі даного класу, за допомогою методів, що реалізують алгоритми генерації значень для цих масивів.

В класі ODCMEncoder реалізовано два методи, які відповідно здійснюють шифрування та розшифрування вхідного масиву байт.

Шифрування здійснюється за допомогою наступного методу:

```
public int[] encode(int[] data) {
    int r0 = this.r0;
    int h0 = this.h0;
    int[] temp = new int[data.length];
    for (int i = 0; i < data.length; i++) {
        temp[i] = r0;
        r0 = cm.getMr()[r0][data[i]];
    }
    for (int i = data.length - 1; i >= 0; i--) {
        temp[i] = cm.getM()[h0][temp[i]][data[i]];
        h0 = cm.getMh()[h0][data[i]];
    }
    return temp;
}
```

Де temp – допоміжний масив, this.r0 та this.h0 – початкові значення для бокових функцій ОККМ, а cm – об’єкт класу CMData, що містить всі необхідні дані для КМ.

Розшифрування здійснюється за допомогою наступного методу:

```
public int[] decode(int[] data){
    int r0 = this.r0;
    int h0 = this.h0;
    int[] temp = new int[data.length];
    for (int i = 0; i < data.length; i++) {
        temp[i] = r0;
        r0 = cm.getMOr()[r0][data[i]];
    }
```

```
        for (int i = data.length - 1; i >= 0; i--) {
            temp[i] = cm.getMO()[h0][temp[i]][data[i]];
            h0 = cm.getMOh()[h0][data[i]];
        }
        return temp;
    }
```

Клас EncodedText – є типовою сутністю, що містить наступні поля:

```
private int id;
private byte[] encodingKey;
private byte[] encodedText;
```

Де id – поле, що містить унікальний ідентифікатор, конкретного рядку в базі даних, encodingKey – масив байт, що містить в собі ключ, за допомогою якого було зашифровано повідомлення, а encodedText – масив байт, в якою знаходиться зашифроване вхідне повідомлення.

Відповідно об’єкт даного класу використовується для повідомлення, що було зашифровано за допомогою лише одного ОККМ.

Відповідно клас EncodedEncoding – є сутністю, що використовується для повідомлення, що було зашифровано за допомогою суперпозиції двох підстановок, та має наступні поля:

```
private int id;
private byte[] encodingKey;
private byte[] encodingEncodeKey;
private byte[] encodedText;
```

Де id – поле, що містить унікальний ідентифікатор, конкретного рядку в базі даних, encodingKey – ключ першої підстановки, encodingEncodeKey – ключ другої підстановки, encodedText – зашифроване за допомогою суперпозиції цих двох підстановок повідомлення.

Класи EncodedTextDao та EncodedEncodingDao здійснюють запис в базу даних відповідних сутностей EncodedText та EncodedEncoding.

Клас DBConnectionUtil є допоміжним класом, що створює підключення до бази даних та видає його клієнту.

Клас ArraysConverter теж є допоміжним класом, який містить два методи, що здійснюють перетворення масиву int[] в масив byte[] і навпаки. Це необхідно, оскільки, при зчитуванні з файлу байтів, вони помістяться в масив byte[], але так як примітивний тип byte в Java може приймати значення з множити {-127…128}, то необхідно перетворити такий масив в масив int[].

Клас EncodingService здійснює генерацію зашифрованих текстів та записує їх в базу даних, за допомогою вище вказаних класів.

Для генерації зашифрованого повідомлення на базі одного ОККМ використовується наступний метод:

```
public void generateEncodedText() {
    ODCMEncoder encoder = new ODCMEncoder();
    EncodedTextDao encodedTextDao = new EncodedTextDao();
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            encoder.setH0(i);
            encoder.setR0(j);
            EncodedText encodedText = new EncodedText();
            byte[] bytes = toByte(encoder.encode(file));
            encodedText.setEncodingKey(
                    getEncodingKey(encoder.getCm(), encoder.getH0(),
encoder.getR0())
            );
            encodedText.setEncodedText(bytes);
            try {
                encodedTextDao.addEncodedText(encodedText);
```

```
            } catch (SQLException e) {
                e.printStackTrace();
                System.out.println(Arrays.toString(bytes));
            } catch (ClassNotFoundException e) {
                e.printStackTrace();}}}}
```

Для генерації зашифрованого повідомлення на базі двох ОККМ використовується наступний метод:

```
    public void generateEncodedEncodes() {
        internalEncoder.setCm(new CMData(8));
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                internalEncoder.setH0(i);
                internalEncoder.setR0(j);
                encodedEncoding.setEncodingKey(
    getEncodingKey(internalEncoder.getCm(), internalEncoder.getH0(),
    internalEncoder.getR0())
                );
                int[] encodedData = internalEncoder.encode(file);
                externalEncoder.setCm(new CMData(8));
                for (int k = 0; k < 2; k++) {
                    for (int l = 0; l < 2; l++) {
                        externalEncoder.setH0(k);
                        externalEncoder.setR0(l);
                        encodedEncoding.setEncodingEncodeKey(
    getEncodingKey(externalEncoder.getCm(), externalEncoder.getH0(),
    externalEncoder.getR0()));
    encodedEncoding.setEncodedText(toByte(externalEncoder.encode(encoded
    Data)));
                        try {
```

```
                encodedEncodingDao.addEncodedEncoding(encodedEncoding);
            } catch (SQLException | ClassNotFoundException e) {
                e.printStackTrace();}}}}}}
```

В результаті, було згенеровано сто мільйонів різних шифрограм для одного і того ж повідомлення, за допомогою двох варіантів алгоритму шифрування: з одним ОККМ та з суперпозицією підстановок двох ОККМ,. В результаті, не було виявлено жодної колізії між утвореними обома варіантами шифрограмами. Отже, давати точні твердження важко, але можна зробити висновок, що суперпозиція підстановок даного алгоритму буде дорівнювати одній підстановці є малоймовірною.

Висновки до розділу 2

В розділі 1 було визначено два види симетричних алгоритмів шифрування: потоковий, блочний. Але алгоритм шифрування на базі лінійних структур важко віднести, до одного з цим видів. В такому разі пропонується віднести даний алгоритм, до виду, який можна назвати, блочно-поточні алгоритми. Оскільки даний алгоритм можна налаштувати на роботу як і поточного алгоритму, так і блочного.

Для того, щоб він працював, наприклад, у блочному режимі, потрібно лише розбити відкритий текст на блоки визначеного наперед розміру. Тоді кожен такий блок буде шифруватися за допомогою окремого ОККМ. Також при даній реалізації не виникне проблема визначення останнього блоку повідомлення, якщо розмір відкритого тексту не буде кратний розміру блоку. Оскільки розмір шифротексту дорівнює розміру відкритого тексту та при використанні 8-розрядних підстановок не потрібно задумуватися над розміром блоку який був би кратним розміру підстановки. В такому випадку на стороні приймача, при заданому наперед розміру блоку, не буде проблемою визначити розмір останнього блоку. Також можна пришвидшити виконання шифрування, якщо шифрування кожного блоку запустити в

окремому потоці. Для цього потрібно буде лише наперед визначити таблиці підстановок в КМ на базі яких буде здійснюватися шифрування.

Отже, можна дійти висновку, що алгоритми криптографічних перетворень на базі лінійних структур мають широкі можливості по своїй реалізації. Оскільки, використовуючи базовий алгоритм шифрування на базі підстановок довільної розрядності, з легкістю можна робити до нього модифікації для виконання поставлених задач. Наприклад, можна з легкістю змінити розмір підстановок в КМ. Також за рахунок того, що шифрування та розшифрування відбувається за допомогою одного і того ж алгоритму, це дозволяє без особливих проблем здійснити апаратну реалізацію.

Постає наступна проблема – що відносити до секретних даних алгоритму.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ СПОСОБІВ ГЕНЕРАЦІЇ КЛЮЧІВ АЛГОРИТМУ ШИФРУВАННЯ

3.1 Визначення секретних даних

Головним елементом, при здійсненні шифрування та розшифрування, в КМ є таблиці підстановок, оскільки навіть формування таблиць вибору підстановок здійснюється на основі відповідних пар таблиць підстановок. В такому випадку секретними даними можна вважати таблиці підстановок КМ.

Оскільки в сучасному світі розміри файлів кратні 8 бітам, то для шифрування доцільно використовувати 8-розрядні підстановки. Тоді розмір таблиці підстановок дорівнює 2^{11} біт, а розмір всіх чотирьох таблиць – 2^{13} біт. Також необхідно додати два біти початкових значень h та r, що подаються відповідно на перший та останній КМ. Але оскільки таблиці підстановок достатньо визначити лише для 255 значень вхідного аргументу X, то 256 – те значення, що визначається за відсутністю в 255 значеннях таблиці, що є достатнім, щоб цих два біти не враховувати. В такому разі розмір ключа буде мати розмір 1КБайт [22].

Якщо використовувати модифікований алгоритм генерації таблиць вибору підстановок і в них генерується велика кількість класів, то в якості секретного ключа можна використовувати таблиці вибору підстановок та порядок таблиць підстановок. Тоді таблиці підстановок можна вважати публічними.

Модифікований алгоритм генерації таблиць вибору підстановок було застосовано для аналізу 4-х таблиць підстановок, що використовуються в алгоритмі шифрування «Калина». В результаті було утворену наступну множину таблиць вибору підстановок з визначеною кількість класів, в залежності від пар таблиць підстановок (табл. 3.1):

Таблиця 3.1 – Кількість класів в залежності від пар підстановок

Пари підстановок	π_0-π_3	π_0-π_2	π_0-π_1	π_1-π_3	π_1-π_2	π_2-π_3
Кількість класів	2	6	8	12	8	8

Таким чином, може бути утворено 24 різних КМ, в залежності від порядку таблиць підстановок в КМ. З них можна виділити три різних за кількістю можливих згенерованих таблиць вибору підстановок КМ, в залежності від пар таблиць підстановок (табл. 3.2).

Таблиця 3.2 - Групи КМ за парами підстановок та з кількістю можливих згенерованих таблиць вибору підстановок

Пари підстановок	Кількість можливих таблиць вибору підстановок	Пари підстановок	Кількість можливих таблиць вибору підстановок	Пари підстановок	Кількість можливих таблиць вибору підстановок
π_1-π_2	254	π_1-π_2	254	π_1-π_3	4094
π_0-π_3	2	π_0-π_3	2	π_0-π_2	62
π_0-π_1	254	π_1-π_3	4094	π_0-π_1	254
π_2-π_3	254	π_0-π_2	62	π_2-π_3	254

В результаті кількість можливих згенерованих таблиць вибору підстановок дорівнює 2^{37}. В такому випадку, секретний ключ буде мати розрядність 2^{42} і буде складатися з таблиць вибору підстановок та порядку таблиць підстановок.

Шляхом використання операції множення в групі підстановок, створених із *t* ОККМ на базі S-box Калини, розмір ключа буде дорівнювати

42t. Таким чином вже при t=3 розмір ключа буде становити 126 біт, що є досить високим показником у сучасній криптографії [16].

«При використанні алгоритму шифрування на базі лінійних структур з використанням суперпозиції підстановок необхідно створити множину **Q** КМ, яка не є секретною. В такому випадку шифрування та розшифрування здійснюється за допомогою підстановки, яка є суперпозицією підстановок із випадково вибраних КМ. Тоді секретним ключем буде послідовний перелік обраних КМ із множини Q, кількість таких КМ та початкові значення h та r для кожного ОККМ. Розмір ключа обчислюється за наступною формулою:

$$K = log_2|Q| + 2q + qlog_2|Q|,$$

де q – кількість обраних КМ [6].

Тоді, якщо при реалізації даного алгоритму використовувати модифікований алгоритм генерації таблиць вибору підстановок, то розмір ключа буде дорівнювати $q \times k$, де q – кількість обраних КМ, а k – розмір ключа для кожного ОККМ, де ключем є таблиці вибору підстановок та порядок таблиць підстановок» [22].

3.2 Класифікація силових атак

Головною метою криптоаналітика є злом криптосистеми та визначення відкритого тексту з зашифрованого тексту. Щоб отримати відкритий текст, зловмисник повинен лише з'ясувати секретний ключ розшифровки, оскільки алгоритм вже знаходиться у відкритому доступі.

Таким чином, він прикладає максимум зусиль для з'ясування секретного ключа, який використовується в криптосистемі. Як тільки зловмисник може визначити ключ, атакована система вважається зламаною або скомпрометованою.

На основі методології, що використовується, атаки на криптосистеми можна поділити на кілька груп таким чином:

1. Ciphertext Only Attacks (COA) - у цьому методі криптоаналітик має доступ до набору зашифрованих текстів. Він не має доступу до відповідного відкритого тексту. Кажуть, що COA, є успішним, якщо відповідний відкритий текст може бути визначений з заданого набору зашифрованого тексту. Інколи за допомогою цієї атаки може бути визначений ключ шифрування.

2. Known Plaintext Attack (KPA) - у цьому методі атакуючий знає відкритий текст для деяких частин зашифрованого тексту. Завдання полягає в тому, щоб розшифрувати іншу частину зашифрованого тексту за допомогою цієї інформації. Це може бути зроблено шляхом визначення ключа або використання іншого методу.

3. Chosen Plaintext Attack (CPA) - у цьому методі криптоаналітик має зашифрований вибраний текст. Таким чином, у нього є пара зашифрованого тексту та відкритого тексту за його вибором. Це спрощує його завдання визначити ключ шифрування. Прикладом цього нападу є диференціальний криптоаналіз, який застосовується проти блочних шифрів, а також хеш-функцій. Популярна криптосистема з відкритим ключем, RSA також є вразливою до атак даного типу [23].

«Для атак першої групи для блочних криптографічних перетворень існує метод Шеннона по визначенню ненадійності ключів, який ґрунтується на наявності правил створення початкових повідомлень. Метод дозволяє з кожним наступним блоком криптограми відкидати частину ключів. У випадку, де ключ складається з таблиць підстановок, цей метод не можна застосувати, оскільки при додаванні кожного наступного байту початкового повідомлення змінюються всі попередні байти криптограми. Необхідно відмітити, що в режимах шифрів Калина та AES, зміна блоку приводить до змін шифрограм лише наступних блоків, а не попередніх.

У випадку атак другої групи крипто аналітик в змозі визначити декілька значень підстановок ключа, але не підстановки в цілому. А при

методі силової атаки необхідно проаналізувати 4(w+1) підстановок, де w=(2m)! , які реалізуються всіма можливими каскадами при будь яких h0 та r0 , що вже при m=3 практично не можливо.

У випадку атак третьої групи криптоаналітик в змозі задати всі 16-розрядні початкові дані (криптограми) та визначити всі чотири прямі (або обернені) підстановки, і тим самим повністю визначити ключ. Отже, алгоритм шифрування, який ґрунтується на одному каскаді можна вважати стійким лише для атак першої та другої групи. Але для запобігання атак із третьої групи достатньо в сервісі криптографічних перетворень аналізувати появу численних вхідних даних, які відрізняються лише в декількох байтах, та блокувати, в цьому випадку, роботу сервісу.

Тоді при використанні алгоритму шифрування на базі лінійних структур з використанням суперпозиції підстановок результати аналізу атаки на ключ в загалом відповідають результатам випадку, де ключ складається з таблиць підстановок, але необхідно відмітити наступні особливості. Силова атака на ключ відповідає перебору всіх 2^K ключів як і в стандартних алгоритмах, але при цьому збільшується обсяг обчислень пов'язаний з визначенням 2q таблиць вибору підстановок. В сервісах шифрування та розшифрування таблиці вибору підстановок можуть бути визначені заздалегідь» [22].

3.3 Генерація таблиць підстановок

«Таблиці підстановок алгоритму шифрування на базі лінійних структур задаються випадковим чином. В такому випадку для генерації чисел, необхідно використати криптографічно стійкий генератор псевдовипадкових чисел, оскільки якщо криптоаналітик зможе визначити початковий стан генератора, то він буде в змозі визначити таблиці підстановок, а в такому випадку, він може розшифрувати всі перехоплені криптограми. Саме тому такий генератор повинен відповідати наступним вимогам:

1. Задовольняти «тест на наступний біт». Суть тесту полягає в наступному: не повинно існувати поліноміального алгоритму, який знаючи перші k біт випадкової послідовності може передбачити k+1 біт із ймовірністю більше 50%;

2. Залишатися надійним навіть у випадку, якщо частина або всі його початкові стани стали відомими криптоаналітику.

Головним міжнародним стандартом, що стандартизує алгоритми генерації послідовностей випадкових чисел, є міжнародний стандарт ISO/IEC 18031 «Information technology — Random number generation», який визначає алгоритми генерації псевдовипадкових і випадкових чисел, а також визначає статистичні тести перевірки генераторів.

Прикладами таких генераторів є спеціальний файл ОС Unix /dev/random, реалізований в Linux або функція CryptGetRandom представлена в CryptoAPI компанії Microsoft» [16].

Оскільки, якщо генерувати значення в таблицях підстановок по одній, то велика кількість значень, що видасть генератор випадкових чисел, будуть дублюватися з уже існуючими значеннями в даній таблиці. Тому було запропоновано алгоритм генерацій значень в таблицях підстановок, «який має наступну схему:

1) кожна таблиця підстановок має свій лічильник з початковим значенням 0;

2) генератор випадкових чисел видає нове число;

3) елемент таблиці підстановок з індексом, що дорівнює числу, яке видав генератор, перевіряється на пустоту;

4) якщо елемент пустий, то лічильник даного масиву збільшується на 1 і нове значення лічильника записується в даний елемент масиву;

5) якщо елемент вже має значення, то п.п. 3-5 повторюються для наступної таблиці;

6) якщо таблиць більше не залишилося, то виконується перевірка на те чи всі таблиці заповнені;

7) якщо всі таблиці заповнені, то виконання алгоритму завершується, інакше алгоритм продовжує виконуватися починаючи з п. 2» [16].

Алгоритм генерації таблиць підстановок реалізований мовою Java має наступний вигляд:

```
Arrays.fill(M[0][0], -1);
Arrays.fill(M[0][1], -1);
Arrays.fill(M[1][0], -1);
Arrays.fill(M[1][1], -1);
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
int[][] MI = new int[][]{{0, 0}, {0, 0}};
int index;
while (true) {
   index = random.nextInt(256);
   for (int i = 0; i < 2; i++) {
      for (int j = 0; j < 2; j++) {
         if (M[i][j][index] == -1) {
            M[i][j][index] = MI[i][j];
            MI[i][j]++;
         }}}
   if(isTablesFilled()){
      break;
   }
}
```

Де M – тривмірний масив, який містить в собі масиви підстановок.

Метод isTablesFilled(), перевіряє чи вже повністю заповнені масиви підстановок і має наступну реалізацію:

```
private boolean isTablesFilled() {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < M[i][j].length; k++) {
                if(M[i][j][k] == -1){
                    return false;
                }}}}
    return true;
}
```

Клас SecureRandom дозволяє генерувати випадкові числа, за допомогою використання криптостійкого генератора випадкових чисел, що представлений функцією CryptGetRandom в CryptoAPI, що представлена в операційній системі Windows.

3.4 Модифікований алгоритм генерації таблиць підстановок

«Суть модифікації для 4 таблиць підстановок полягає в наступному. Перша таблиця підстановок генерується згідно з алгоритмом, а всі наступні, генеруються таким чином, щоб кількість елементів в класах таблиць вибору підстановок була не більшою за 4» [16].

Даний алгоритм можна поділити на 3 етапи:

1. Генерація першої таблиці;
2. Генерація на базі першої таблиці інших двох таблиць, що є в парі з нею при генерації таблиць вибору підстановок;
3. Генерація четвертої таблиці на базі таблиць згенерованих на етапі 2.

Схема алгоритму для першого етапу наступна:

1. лічильнику для даної таблиці виставляється початкове значення 0;
2. криптостійкий генератор випадкових чисел видає нове число;
3. якщо елемент з індексом, що дорівнює числу видному генератором, вже має якесь значення, то п.п.2-3 виконуються, ще раз;

4. якщо ж елемент пустий, то в нього записується значення лічильника;

5. значення лічильника збільшується на 1;

6. якщо в таблиці більше немає пустих елементів, то виконання алгоритму завершується, інакше алгоритм продовжує виконуватися з п.2.

Програмна реалізація даного етапу алгоритму мовою Java виглядає наступним чином:

```
Arrays.fill(M[0][0], -1);
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
int nextVal = 0;
int index;
while (true) {
   index = random.nextInt(elementsNumber);
   if (M[0][0][index] == -1) {
            M[0][0][index] = nextVal;
            nextVal++;
        }
   if(isTableFilled(M[0][0])){
      break;
   }
}
```

Де M[0][0] – масив, що містить значення першої таблиці підстановок.

Схема алгоритму для другого етапу наступна:

1. задається лічильник *i* з початковим значення рівним 0;

2. якщо значення лічильнику *i* більше довжини масиву підстановок, то алгоритм завершує своє виконання;

3. дістається значення елементу масиву підстановок з індексом *i*;

4. якщо значення елементу не пусте, то значення лічильника *i* збільшується на 1 і п.2 виконується ще раз;

5. якщо ж елемент пустий, то алгоритм продовжує виконуватися з п. 5;

6. значення елементу масиву першої таблиці підстановок з індексом *i* записується у спеціальну змінну *m*;

7. значення індексу елементу записується в змінну *j*;

8. значення кількості елементів в класі записується у змінну, яка буде зберігати скільки елементів, ще необхідно додати, щоб клас був повним;

9. генератор випадкових чисел видає нове число;

10. якщо значення числа дорівнює значенню індексу в змінній *j* або елемент масиву підстановок з індексом, що дорівнює згенерованому числу, не пустий, то п.9 виконується ще раз.

11. якщо елемент пустий і індекси різні, то елементу присвоюється значення змінної *m*;

12. змінній *m* присвоюється значення елементу масиву першої таблиці підстановок з індексом, що дорівнює згенерованому числу;

13. значення кількості елементів, що необхідно додати до класу, зменшується на 1;

14. якщо значення кількості елементів дорівнює 1, то елементу масиву з індексом *i*, присвоюється значення *m*, значення лічильника збільшується на 1 і алгоритм продовжує виконуватися з п.2.

Програмна реалізація даного алгоритму мовою Java має наступний вигляд:

```
for (int i = 0; i < M[0][0].length; i++) {
   if (M[0][1][i] == -1) {
      int val = M[0][0][i];
      int index = i;
      int elementsLeft = elementsInClass;
      while (true) {
         int randomIndex = random.nextInt(M[0][1].length);
```

```
if (elementsLeft == 1) {
   M[0][1][index] = val;
   break;
}
if (randomIndex != index && M[0][1][randomIndex] == -1) {
   M[0][1][randomIndex] = val;
   val = M[0][0][randomIndex];
   elementsLeft--;}}}}
```

Необхідно зазначити, що перед виконання алгоритму, потрібно визначити зі скількох елементів буде складатися клас в таблицях вибору підстановок.

Аналогічно алгоритм виконується і для третьої таблиці підстановок, оскільки, як і друга, вона генерується на базі першої таблиці. Таким чином за допомогою цього алгоритму вже генеруються дві пари таблиць підстановок, на базі яких можуть згенеруватися дві відповідні таблиці вибору підстановок визначеною кількістю елементів в класах.

Отже, залишається лише згенерувати останню четверту таблицю підстановок, яка дозволить генерувати ще дві таблиці вибору підстановок.

Схема алгоритму для третього етапу наступна:

1. задається лічильник *i* з початковим значення рівним 0;
2. якщо значення лічильнику *i* більше довжини масиву підстановок, то виконується п.20;
3. дістається значення елементу масиву з індексом *i*;
4. якщо значення елементу не пусте, то значення лічильника *i* збільшується на 1 і п.2 виконується ще раз;
5. якщо ж елемент пустий, то алгоритм продовжує виконуватися з п. 5;
6. значення елементу масиву другої таблиці підстановок (M_{01}) з індексом *i* записується у спеціальну змінну *m*;

7. значення індексу елементу записується в змінну *j*;

8. значення кількості елементів в класі записується у змінну, яка буде зберігати скільки елементів, ще необхідно додати, щоб клас був повним;

9. якщо елемента зі значенням, що дорівнює змінній *m*, немає в масиві, то алгоритм продовжує виконуватися з п. 20;

10. якщо значення кількості елементів більше одиниці, то виконується п.12;

11. якщо значення кількості елементів менше або дорівнює одиниці, то елемент в який було останнім записане якесь значення знову робиться пустим, а кількість елементів повертається до позначки, яка була перед записом значення в цей елемент, і алгоритм продовжує виконуватися з п.14;

12. кількість елементів в класі зменшується на 1, а змінній *m* присвоюється значення елементу масиву M_{01} з індексом елементу масиву M_{11}, що дорівнює попередньому значенню *m*;

13. п.п.18-12 виконуються доки є в масиві M_{11} є елементи, що дорівнюють *m*;

14. генератор випадкових чисел видає нове число;

15. якщо значення числа дорівнює значенню індексу в змінній *j* або елемент масиву підстановок з індексом, що дорівнює згенеровано числу, не пустий, то п.14 виконується ще раз.

16. якщо елемент пустий і індекси різні, то елементу присвоюється значення змінної *m*;

17. змінній *m* присвоюється значення елементу масиву першої таблиці підстановок з індексом, що дорівнює згенерованому числу;

18. значення кількості елементів, що необхідно додати до класу, зменшується на 1;

19. якщо значення кількості елементів дорівнює 1, то елементу масиву з індексом *i*, присвоюється значення *m*, значення лічильника збільшується на 1 і алгоритм продовжує виконуватися з п.2;

20. п.п.1-19 виконуються, ще раз тільки замість масиву M_{01} буде масив третьої таблиці підстановок M_{10}, якщо алгоритм виконався і для цього масиву, то виконання продовжується з п.21;

21. для масивів двох згенерованих масивів знаходяться елементи з однаковими значеннями і записуються в допоміжний масив такого ж розміру, як і масиви підстановок, з такими ж індексами, як в оригінальних масивах;

22. масиви знову стають пустими, але при цьому в них записуються значення елементів з допоміжного масиву з такими ж індексами;

23. п.п.1-22 виконуються до тих пір, поки кількість спільних елементів у згенерованих масивах не буде дорівнювати розміру масиву підстановок четвертої таблиці;

24. після цього спільні елементи записуються в масив M_{11}.

В результаті виконання всіх трьох етапів буде згенеровано всі чотири таблиці підстановок на базі яких можуть бути утворені таблиці вибору підстановок із визначеною кількістю елементів у класах.

«В такому випадку, при розрядності підстановок рівній 8, кількість класів згенерованих у відповідних таблицях вибору підстановок буде не менша за 64. Тоді може бути згенеровано мінімум 2^{256} різних таблиць вибору підстановок, а, відповідно, розрядність ключа буде дорівнювати 256. Це забезпечує можливість вважати самі підстановки не секретними і суттєво зменшити розмір ключа для алгоритму шифрування, що базується одному каскаді» [16].

Висновки до розділу 3

Визначено, що у випадку з використання алгоритму шифрування, що базується на одному каскаді, до секретних даних можна віднести таблиці підстановок.

Також є варіант з використанням таблиць вибору підстановок, як ключа шифрування, але для цього потрібно генерувати таблиці підстановок

за допомогою модифікованого алгоритму. Для того, щоб збільшити кількість класів і відповідно збільшити розрядність секретного ключа.

Було поставлено експеримент, з використанням таблиць підстановок шифру «Калина». З метою можливого їх використанні у алгоритмі шифрування на базі лінійних структур. Та доведено, що розрядність ключа шифрування може буті не меншою за 128 розрядів, при використанні операції множення в групі підстановок [16].

Проведено теоретичний аналіз можливих криптоатак на ключ. Та з'ясовано, що ключ алгоритму шифрування на базі одного каскаду є стійкий до COA та KPA атак, але може бути повністю скомпрометованим при CPA атаці. В результаті визначено, що для протистояння CPA атакам, можна згенерувати множину КМ. В такому випадку шифрування буде відбуватися за рахунок суперпозиції підстановок із множини КМ. Тоді секретним ключем буде вважатися послідовний перелік використаних КМ та їх кількість.

ВИСНОВКИ

Було проведено аналіз та порівняння існуючи стандартів шифрування, а саме розглянуто схему їх роботи, способи генерації ключів та швидкість. Описано алгоритм шифрування на базі регулярних структур та проведено порівняльний аналіз швидкості даного алгоритму зі стандартами шифрування. З отриманих в результаті порівняння результатів можна зробити висновок, що даний алгоритм є на порядок швидшим, за рахунок того, що виконує меншу кількість операцій при шифруванні. Особливо це помітно при шифруванні великих обсягів даних.

Також, оскільки алгоритм на базі регулярних структур є дуже гнучким, було запропоновано кілька його можливих реалізацій. Так як алгоритм шифрування є ідентичним алгоритму розшифрування, з тією різницею, що для розшифрування необхідно використовувати обернені КМ, що формуються на базі звичайного КМ, то було запропоновано апаратну реалізацію на базі ПЛІС. А конкретніше для формування КМ пропонується використовувати так звані LUT. Таким чином за допомогою апаратної реалізації можна організовувати 2- та 4-розрядні підстановки. Також було представлено програмну реалізацію даного алгоритму, де було вказано алгоритм генерації таблиць підстановок та таблиць вибору підстановок в КМ, генерації оберненого КМ та сам алгоритм шифрування.

В результаті було зроблено висновок, що даний алгоритм не можна віднести ні до поточних, ні до блочних. Таким чином було запропоновано віднести його до нового виду блочно-поточних алгоритмів. Оскільки його можна налаштувати щоб він працював як в поточному режимі, так і в блочному. При використанні блочного режиму можна пришвидшити шифрування за рахунок виконання шифрування блоків в окремих потоках.

Було проаналізовано, які дані алгоритму можна віднести до секретних та теоретично проаналізовано можливі атаки на ключ. В результаті зроблено

висновок, що при використанні лише одного каскаду КМ для шифрування, то секретний ключ стійкий лише до COA та KPA атак. При цьому до секретних даних можна віднести чотири таблиці підстановок, тоді розмір ключа, при використанні 8-розрядних підстановок, буде мати розмір 1КБайт. Ще одним варіантом ключа можуть бути таблиці вибору підстановок, за умови великої кількісті класів в цих таблицях В такому випадку таблиці підстановок можна вважати публічними. Для забезпечення генерації великої кількості класів у таблицях вибору підстановок було запропоновано модифікований алгоритм генерації цих таблиць.

Для того, щоб ключ був стійким ще й до CPA атак, можна створити множину КМ. В такому випадку шифрування буде відбуватися за рахунок суперпозиції підстановок з випадково вибраних КМ зі створеної попередньо множини. Тоді ключем шифрування буде послідовний перелік обраних КМ та їх кількість.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Хто володіє інформацією - той володіє світом [Електронний ресурс]. – 2017. – Режим доступу: *https://nnovosti.info/articles/hto_volodije_informatsijeju_toj_volodije_svitom-622.html*

2. 5G [Електронний ресурс]. – 2018. – Режим доступу: *https://uk.wikipedia.org/wiki/5G*

3. Шифр Цезаря [Електронний ресурс]. – 2017. – Режим доступу: *http://ru.science.wikia.com/wiki/Шифр_Цезаря*

4. В1949 году Клод Шеннон опубликовал статью "Теория связи в секретных системах", поставившую криптографию в ранг науки. [Електронний ресурс]. – 2010. – Режим доступу: *https://www.securitylab.ru/informer/240627.php*

5. Криптографія: загальні визначення, класифікація.асиметричні та симетричні криптоалгоритми, їх порівняння. [Електронний ресурс]. – 2014. – Режим доступу: *https://dehtyarov09.wordpress.com/2014/03/16/криптографія-загальні-визначення-кл-2/*

6. Криптографические методы защиты информации. [Електронний ресурс]. – 2014. – Режим доступу: *http://www.volpi.ru/umkd/zki/index.php?man=1&page=15*

7. Блочные шифры. [Електронний ресурс]. – 2011. – Режим доступу: *http://kryptography.narod.ru/block.html*

8. Feistel Ciphers (or Feistel Network) [Електронний ресурс]. – 2017. – Режим доступу: *https://www.commonlounge.com/discussion/df78c412191849029996f37b1089f3a4*

9. The DES Algorithm Illustrated [Електронний ресурс]. – 2006. – Режим доступу: *http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm*

10. A Detailed Description of DES and 3DES Algorithms (Data Encryption Standard and Triple DES) [Електронний ресурс]. – 2018. – Режим доступу: *https://www.commonlounge.com/discussion/5c7c2828bf6b4724b806a9013a5a4b99*

11. The Advanced Encryption Standard (AES) Algorithm [Електронний ресурс]. – 2018. – Режим доступу: *https://www.commonlounge.com/discussion/e32fdd267aaa4240a4464723bc74d0a5*

12. Усовершенствованный стандарт шифрования (AES — Advanced Encryption Standard) [Електронний ресурс]. – 2013. – Режим доступу: *https://www.intuit.ru/studies/courses/552/408/lecture/9365*

13. A New Encryption Standard of Ukraine: The Kalyna Block Cipher [Електронний ресурс]. – 2014. – Режим доступу: *https://eprint.iacr.org/2015/650.pdf*

14. RSA АЛГОРИТМ [Електронний ресурс]. – 2014. – Режим доступу: *https://kovalchukmm14.wordpress.com/2014/12/16/ rsa-алгоритм/*

15. *Невмержицький П.А., Тесленко О.К.* "Алгоритм шифрування, який ґрунтується на суперпозиції підстановок із найпростіших регулярних ОККМ" Свідоцтво про реєстрацію авторського права на твір. № 66618 Дата реєстрації 13.07.2016

16. *Тесленко О.К., Кісільчук Б.Я.* ФОРМУВАННЯ КЛЮЧІВ АЛГОРИТМУ ШИФРУВАННЯ НА БАЗІ ЛІНІЙНИХ СТРУКТУР // Збірник тез доповідей одинадцятої наукової конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2018-2), Київ, НТУУ «КПІ», 14-16 листопада 2018 р.

17. *Тесленко О.К., Невмержицький П.А.* ЗАГАЛЬНА ОЦІНКА АЛГОРИТМУ ШИФРУВАННЯ, ЯКИЙ ГРУНТУЄТЬСЯ НА СУПЕРПОЗИЦІЇ ПІДСТАНОВОК ІЗ НАЙПРОСТІШИХ РЕГУЛЯРНИХ

ОККМ // Збірник тез доповідей восьмої наукової конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2016), Київ, НТУУ «КПІ», 20-25 квітня 2016 р-с.159-164

18. Я.Р. Совин, В.І. Отенко, Є.Ф. Штефанюк. ЕФЕКТИВНА РЕАЛІЗАЦІЯ АЛГОРИТМУ БЛОКОВОГО СИМЕТРИЧНОГО ШИФРУВАННЯ ДСТУ 7624:2014 («КАЛИНА») ДЛЯ 8/16/32-БІТОВИХ ВБУДОВАНИХ СИСТЕМ

19. *В.П. Тарасенко, О.К. Тесленко, О.Ю. Яновська.* Реалізація повних підстановок на простому двомодульному каскаді конструктивних модулів. Інформаційні технології та комп'ютерна інженерія.№1(11)2008. – С.88-97

20. *Тарасенко В.П., Тесленко О.К., Яновська О.Ю.* Реалізація обернених підстановок на простому двомодульному каскаді конструктивних модулів// Інформаційні технології та комп'ютерна інженерія 2013, No3 (28). – С. 16-25.

21. ПЛІС. [Електронний ресурс]. – 2018. – Режим доступу: https://uk.wikipedia.org/wiki/ПЛІС

22. *Кісільчук Б.Я., Тесленко О.К.* Ключі алгоритму шифрування на базі підстановок довільної розрядності // Системний аналіз та інформаційні технології: матеріали 20-ї Міжнародної науково-технічної конференції SAIT 2018, 2018. – С. 230, 231.

23. Attacks On Cryptosystems [Електронний ресурс]. – 2015. – Режим доступу:*https://www.tutorialspoint.com/cryptography/attacks_on_cryptosystems.htm*